ISO/IEC JTC 1/SC 22 N 4456

2009-06-30

ISO/IEC JTC 1/SC 22
Programming Languages

Document Type:	**Text for CD Ballot**
Document Title:	**Combined Registration and CD Ballot for ISO/IEC 15291, Information technology – Programming languages – Ada Semantic Interface Specification (ASIS)**
Document Source:	**WG 9 Convener (J. Tokar)**
Document Status:	**This document is circulated for concurrent CD Registration and CD Ballot. Please submit your vote via the online balloting system by the due date indicated.**
Action ID:	**VOTE**
Due Date:	**2009-10-01**
No. of Pages:	**369**

Secretariat, ISO/IEC JTC 1, American National Standards Institute, 25 West 43rd Street, New York, NY 10036; Telephone: 1 212 642 4932; Facsimile: 1 212 840 2298; Email: lrajchel@ansi.org

ISO/IEC JTC1/SC 22 N

Date: 2009-05-15

ISO/IEC CD 15291

ISO/IEC JTC 1/SC 22/WG 9

Information technology — Programming languages — Ada Semantic Interface Specification (ASIS)

Technologies de l'information — Langages de programmation — Spécification d'interface pour la sémantique Ada

Revision of first edition (ISO 15291:1999)

Warning

This document is not an ISO International Standard. It is distributed for review and comment. It is subject to change without notice and may not be referred to as an International Standard.

Recipients of this draft are invited to submit, with their comments, notification of any relevant patent rights of which they are aware and to provide supporting documentation.

Copyright notice

Contents

Foreword

ISO (the International Organization for Standardization) and IEC (the International Electrotechnical Commission) form the specialized system for worldwide standardization. National bodies that are members of ISO or IEC participate in the development of International Standards through technical committees established by the respective organization to deal with particular fields of technical activity. ISO and IEC technical committees collaborate in fields of mutual interest. Other international organizations, governmental and non-governmental, in liaison with ISO and IEC, also take part in the work. In the field of information technology, ISO and IEC have established a joint technical committee, ISO/IEC JTC 1.

International Standards are drafted in accordance with the rules given in the ISO/IEC Directives, Part 2.

The main task of the joint technical committee is to prepare International Standards. Draft International Standards adopted by the joint technical committee are circulated to national bodies for voting. Publication as an International Standard requires approval by at least 75% of the national bodies casting a vote.

International Standard ISO/IEC 15291 was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, Subcommittee 22, *Programming languages, their environments and system software interfaces*.

This second edition cancels and replaces the first edition (ISO/IEC 15291:1999), of which it constitutes a technical revision.

Annexes A, B, C, D, and E of this International Standard are for information only. Annex F forms an integral part of the International Standard.

Introduction

The Ada Semantic Interface Specification (ASIS) is an interface between an Ada environment (as defined by ISO/IEC 8652:1995(E)) and any tool requiring information from it. An Ada environment includes valuable semantic and syntactic information. ASIS is an open and published callable interface which gives tool and application developers access to this information. ASIS has been designed to be independent of underlying Ada environment implementations, thus supporting portability of software engineering tools while relieving tool developers from needing to understand the complexities of an Ada environment's proprietary internal representation.

Examples of tools that benefit from the ASIS interface include: automated code monitors, browsers, call tree tools, code reformatters, coding standards compliance tools, correctness verifiers, dependency tree analysis tools, design tools, document generators, metrics tools, quality assessment tools, reverse engineering tools, re-engineering tools, style checkers, test tools, timing estimators, and translators.

The ASIS interface consists of a set of types, subtypes, and subprograms which provide a capability to query the Ada compilation environment for syntactic and semantic information. The ASIS interface can be separated into two related subsystems, the syntactic subsystem and the semantic subsystem. The *syntactic subsystem* provides the ability to traverse a program based on the syntactic (textual) structure of the program, while the *semantic subsystem* provides the ability to traverse the program based on the semantics (meaning) of the program. Interconnections are provided to allow moving seamlessly between the subsystems.

Package **Asis** is the root of the ASIS interface. It contains common types used throughout the ASIS interface. All ASIS subprogram interfaces are provided using child packages. Some child packages also contain type and subtype interfaces local to the child package. The various child packages of ASIS provide queries about the properties of the entities and other elements that make up an Ada program. Queries are provided to learn about declarations, expressions, statements, compilation units, and so on.

Important common abstractions in the syntactic subsystem include Context, Element, and Compilation_Unit. The type Context helps identify the compilation units considered to be analyzable as part of the Ada compilation environment. The type Element is an abstraction of entities within a logical Ada syntax tree. The type Compilation_Unit is an abstraction for Ada compilation units. In addition, there are two sets of enumeration types covering various kinds of elements and units, such as Expression_Kinds. The element kinds are a set of enumeration types providing a mapping to the Ada syntax. The unit kinds are a set of enumeration types describing the various kinds of compilation units.

A semantic entity in the semantic subsystem is represented by a View. Views can represent semantic concepts like Units, Objects, or Statements, whether or not they exist explicitly in the analyzed program. Queries are then provided to further decompose or traverse these views to other related views of the meaning of the program text. In addition, there is a set of child packages that provide routines to transition back and forth between the original syntactic element and the semantic view to obtain these different perspectives on program meaning or content.

Changes in this Edition

This revised International Standard updates the edition from 1999.

The most significant change in this edition was the addition of the semantic subsystem (see above and Section 23). The new concept of views added by the semantic subsystem addresses a significant shortcoming in the 1999 edition of ASIS – ASIS elements represent entities that do not have explicit declarations (such as classwide types, inherited subprograms, and the like) poorly or not at all. The new concept also simplifies access to properties that are not easily

determined from the syntax of an entity. For instance, a type can become tagged explicitly, by being an interface, via inheritance in a derived type, or by having a progenitor. Thus querying Is_Tagged on an appropriate view is much simpler than analyzing all of those possibilities.

A variety of other changes were made to ASIS to support the 2007 Amendment to Ada [ISO/IEC 8652:1995/AMD 1:2007(E)]. Some of the more significant changes were:

- Added support for newly added constructs to ASIS, including tagged incomplete types, null procedures, overriding indicators, extended returns, exception messages in raise statements, and new predefined pragmas;
- Adjusted ASIS to properly support the object-oriented prefix notation for subprograms;
- Changed ASIS to be able to support the new aggregate features such as defaulted elements; and
- Changed ASIS to use UTF-16 representations, in order that the full character set of an Ada source program can be represented.

In addition, many other changes were made to clarify and improve the existing standard. These should have minimal effect on existing ASIS programs, but will be an aid in creating new programs using the ASIS package interfaces. Important changes include:

- Added a query to determine directly if an element representing a name represents an implicit dereference;
- Replaced traits with a set of direct query functions; and
- Moved obsolescent types and routines to a new annex, simplifying the presentation to new users and making clear which routines are not intended to be used in new programs.

COMMITTEE DRAFT ISO/IEC 15291 CD

Information technology — Programming languages — Ada Semantic Interface Specification (ASIS)

Section 1: General

1.1 Scope

The Ada Semantic Interface Specification (ASIS) is an interface between an Ada environment (as defined by ISO/IEC 8652:1995(E)) and any tool requiring information from this environment. An Ada environment includes valuable semantic and syntactic information. ASIS is an open and published callable interface which gives tool and application developers access to this information. ASIS has been designed to be independent of underlying Ada environment implementations, thus supporting portability of software engineering tools while relieving tool developers from needing to understand the complexities of an Ada environment's proprietary internal representation.

Examples of tools that benefit from the ASIS interface include: automated code monitors, browsers, call tree tools, code reformatters, coding standards compliance tools, correctness verifiers, dependency tree analysis tools, design tools, document generators, metrics tools, quality assessment tools, reverse engineering tools, re-engineering tools, safety and security tools, style checkers, test tools, timing estimators, and translators.

This International Standard specifies the form and meaning of the ASIS interface to the Ada compilation environment.

This International Standard is applicable to tools and applications needing syntactic and semantic information from the Ada compilation environment.

1.1.1 Extent

This International Standard specifies:

- The form of the ASIS interface;
- Sequencing of ASIS calls;
- The permissible variations within this International Standard, and the manner in which they are to be documented;

- Those violations of this International Standard that a conforming implementation is required to detect, and the effect of attempting to execute a program containing such violations;

This International Standard does not specify:

- Semantics of the interface in the face of simultaneous updates to the Ada compilation environment.
- Semantics of the interface for more than one thread of control.

1.1.2 Structure

This International Standard contains twenty-three sections and six annexes.

Section 1 is general in nature providing the scope of this International Standard, normative references, and definitions.

Section 2 identifies the ASIS technical concepts. Here the Ada compilation environment to which ASIS interfaces is described. The concept of queries is presented. The ASIS package architecture is presented.

The packages that comprise the ASIS International Standard are provided in Sections 3 through 23.

- Section 3 package Asis
- Section 4 package Asis.Errors
- Section 5 package Asis.Exceptions
- Section 6 package Asis.Implementation
- Section 7 package Asis.Implementation.Permissions
- Section 8 package Asis.Ada_Environments
- Section 9 package Asis.Ada_Environments.Containers
- Section 10 package Asis.Compilation_Units
- Section 11 package Asis.Compilation_Units.Times
- Section 12 package Asis.Compilation_Units.Relations
- Section 13 package Asis.Elements
- Section 14 package Asis.Iterator
- Section 15 package Asis.Declarations
- Section 16 package Asis.Definitions
- Section 17 package Asis.Expressions
- Section 18 package Asis.Statements
- Section 19 package Asis.Clauses
- Section 20 package Asis.Text
- Section 21 package Asis.Ids
- Section 22 package Asis.Data_Decomposition (optional package)
- Section 23 package Asis.Views,
  Asis.Program_Units,
  Asis.Subtype_Views,
  Asis.Subtype_Views.Elementary,
  Asis.Subtype_Views.Composite,
  Asis.Object_Views,

Asis.Object_Views.Access_Views,
Asis.Profiles,
Asis.Callable_Views,
Asis.Package_Views,
Asis.Generic_Views,
Asis.Exception_Views,
Asis.Statement_Views,
Asis.Declarations.Views,
Asis.Definitions.Views, and
Asis.Expressions.Views

The following annexes are informative:

- Annex A: Glossary
- Annex B: ASIS Application Examples
- Annex C: Miscellaneous ASIS I/O and IDL Approaches
- Annex D: Rationale
- Annex E: Summary of ASIS Entities

The following annexes are normative:

- Annex F: Obsolescent Features

The major package interfaces visible to ASIS users are identified as clauses facilitating access from the table of contents.

In addition to the basic description of each entity that makes up the ASIS interface, text of various types is labeled with headers.

The various headers and their meaning are:

Implementation Permissions

These items describe permissions given an implementor when implementing the associated type or query.

Implementation Requirements

These items describe additional requirements for conforming implementations.

NOTE These items describe notes of interest to ASIS applications.

Examples

These items give examples of use of ASIS interfaces.

1.1.3 Conformity with this International Standard

1.1.3.1 Implementation conformance requirements

An *ASIS implementation* includes all the hardware and software that implements the ASIS specification for a given Ada implementation and that provides the functionality required by the ASIS specification. An *ASIS implementor* is a company, institution, or other group (such as a vendor) who develops an ASIS implementation. A conforming ASIS implementation shall meet all of the following criteria:

a) The system shall support all required interfaces defined within this International Standard. These interfaces shall support the functional behavior described herein. All interfaces in the ASIS specification are required unless the interface is specifically identified as being optional. The ASIS specification defines one optional package:

Asis.Data_Decomposition. Asis.Data_Decomposition has one child package, Asis.-Data_Decomposition.Portable_Transfer.

b) The system may provide additional facilities not required by this International Standard. *Extensions* are non-standard facilities (e.g., other library units, non-standard children of standard ASIS library units, subprograms, etc.) which provide additional information from ASIS types, or modify the behavior of otherwise standard ASIS facilities to provide alternative or additional functionality. Nonstandard extensions shall be identified as such in the system documentation. Nonstandard extensions, when used by an application, may change the behavior of functions or facilities defined by this International Standard. The conformance document shall define an environment in which an application can be run with the behavior specified by this International Standard. In no case except package name conflicts shall such an environment require modification of a Basic Conforming or Fully Conforming ASIS Application. An implementation shall not add any declarations to the visible part of logical packages defined in the following clauses of this International Standard.

c) An ASIS implementation shall not raise Program_Error on elaboration of an ASIS package, or on execution of an ASIS subprogram, due to elaboration order dependencies in the ASIS implementation.

d) Except as explicitly provided for in this International Standard, Standard.Storage_Error is the only exception that should be raised by operations declared in this International Standard.

e) When executed, an implementation of this International Standard shall not be erroneous, as defined by the Ada Standard.

1.1.3.2 Implementation conformance documentation

A conformance document shall be available for an implementation claiming conformance to this International Standard. The conformance document shall have the same structure as this International Standard, with the information presented in the equivalently numbered clauses, and subclauses. The conformance document shall not contain information about extended facilities or capabilities outside the scope of this International Standard.

The conformance document shall contain a statement that indicates the full name, number, and date of the International Standard that applies. The conformance document may also list software standards approved by ISO/IEC or any ISO/IEC member body that are available for use by a Basic or Fully Conforming ASIS Application. Applicable characteristics whose documentation is required by one of these standards, or by standards of government bodies, may also be included.

The conformance document shall describe the behavior of the implementation for all implementation-defined features defined in this International Standard. This requirement shall be met by listing these features and providing either a specific reference to the system documentation or providing full syntax and semantics of these features. The conformance document shall specify the behavior of the implementation for those features where this International Standard states that implementations may vary.

No specifications other than those described in this subclause shall be present in the conformance document.

The phrase *shall be documented* in this International Standard means that documentation of the feature shall appear in the conformance document, as described previously, unless the system documentation is explicitly mentioned.

The system documentation should also contain the information found in the conformance document.

1.1.3.3 Implementation conformance categories

An implementation is required to define all of the subprograms for all of the operations defined in this International Standard, including those whose implementation is optional. *Required functionality* is the subset of ASIS facilities which are not explicitly identified in the ASIS standard as optional. *Optional functionality* is the subset of ASIS facilities which are explicitly identified in the ASIS standard as optional which may legitimately be omitted from a Basic Conforming ASIS implementation. Optional interfaces shall be included in any Fully Conforming ASIS implementation, unless stated otherwise in the ASIS specification.

If an unimplemented feature is used, the exception Asis.ASIS_Failed shall be raised and Asis.Implementation_Status shall return the value for Error_Kinds of Not_Implemented_Error.

There are four categories of conforming ASIS implementations:

Basic conforming ASIS implementation

A Basic Conforming ASIS Implementation is an ASIS implementation supporting all required interfaces defined within this International Standard.

Fully conforming ASIS implementation

A Fully Conforming ASIS Implementation is an ASIS implementation supporting all required and all optional interfaces defined within this International Standard.

Basic conforming ASIS implementation using extensions

A Basic Conforming ASIS Implementation Using Extensions is an ASIS implementation that differs from a Basic Conforming ASIS Implementation only in that it uses nonstandard extensions that are consistent with this International Standard. Such an implementation shall fully document its extended facilities, in addition to the documentation required for a Basic Conforming ASIS Implementation.

Fully conforming ASIS implementation using extensions

A Fully Conforming ASIS Implementation Using Extensions is an ASIS implementation that differs from a Fully Conforming ASIS Implementation only in that it uses nonstandard extensions that are consistent with this International Standard. Such an implementation shall fully document its extended facilities, in addition to the documentation required for a Fully Conforming ASIS Implementation.

1.1.3.4 Application conformance categories

An *ASIS application* is any programming system or any set of software components making use of ASIS queries to obtain information about any set of Ada components. All ASIS applications claiming conformance to this International Standard shall use a Conforming ASIS Implementation with or without extensions. In any case, an application that accesses an Ada environment directly (other than through ASIS) is not considered to be a conformant application. All Conforming Applications fall within one of the categories defined below.

Basic conforming ASIS application

A Basic Conforming ASIS Application is an application that only uses the required facilities defined within this International Standard. It shall be portable to any Conforming ASIS Implementation.

Fully conforming ASIS application

A Fully Conforming ASIS Application is an application that only uses the required facilities and the optional facilities defined within this International Standard. It shall be portable to any Fully Conforming ASIS Implementation.

Basic conforming ASIS application using extensions

A Basic Conforming ASIS Application Using Extensions is an application that differs from a Basic Conforming ASIS Application only in that it uses nonstandard, implementation provided, extended facilities that are consistent with this International Standard. Such an application should fully document its requirements for these extended facilities. A Basic Conforming ASIS Application Using Extensions may or may not be portable to other Basic or Fully Conforming ASIS Implementation Using Extensions.

Fully conforming ASIS application using extensions

A Fully Conforming ASIS Application Using Extensions is an application that differs from a Fully Conforming ASIS Application only in that it uses nonstandard, implementation provided, extended facilities that are consistent with this International Standard. Such an application should fully document its requirements for these extended facilities. A Fully Conforming ASIS Application Using Extensions may or may not be portable to other Fully Conforming ASIS Implementation Using Extensions.

1.1.4 Classification of errors

ASIS reports all operational errors by raising an exception. Whenever an ASIS implementation raises one of the exceptions declared in package Asis.Exceptions, it will previously have set the values returned by the Status and Diagnosis queries to indicate the cause of the error. The possible values for Status are indicated here along with suggestions for the associated contents of the Diagnosis string.

ASIS applications are encouraged to follow this same convention whenever they explicitly raise any ASIS exception to always record a Status and Diagnosis prior to raising the exception. Values of errors along with their general meanings are:

```
Not_An_Error                -- No error is presently recorded
Value_Error                 -- Routine argument value invalid
Initialization_Error        -- ASIS is uninitialized
Environment_Error           -- ASIS could not initialize
Parameter_Error             -- Bad Parameter given to Initialize
Capacity_Error              -- Implementation overloaded
Name_Error                  -- Context/unit not found
Use_Error                   -- Context/unit not use/open-able
Data_Error                  -- Context/unit bad/invalid/corrupt
Text_Error                  -- The program text cannot be located
Storage_Error               -- Storage_Error suppressed
Obsolete_Reference_Error    -- Semantic reference is obsolete
Unhandled_Exception_Error   -- Unexpected exception suppressed
Not_Implemented_Error       -- Functionality not implemented
Internal_Error              -- Implementation internal failure
```

Diagnostic messages may be more specific.

ASIS defines a set of global exceptions. These exceptions are raised under the circumstances described in 5, “package Asis.Exceptions”

1.2 Normative references

The following standard contains provisions which, through reference in this text, constitute provisions of this International Standard. At the time of publication, the edition indicated was valid. All standards are subject to revision, and parties to agreements based on this standard are encouraged to investigate the possibility of applying the most recent edition of the International Standard indicated below. Members of IEC and ISO maintain registers of currently valid International Standards.

ISO/IEC 8652:1995(E), *Information technology — Programming languages — Ada*.

ISO/IEC 8652:1995/Cor.1:2001(E), *Information technology — Programming languages — Ada* — Technical Corrigendum 1.

ISO/IEC 8652:1995/AMD 1:2007(E), *Information technology — Programming languages — Ada* — Amendment 1.

ISO/IEC 10646:2003, *Information technology — Universal Multiple-Octet Coded Character Set (UCS)*.

1.3 Terms and definitions

The Ada Programming Language is defined by International Standard ISO/IEC 8652:1995(E) as corrected by ISO/IEC 8652:1995/COR1:2001(E) and amended by ISO/IEC 8652:1995/AMD.1:2007(E). This set of documents is collectively known as the *Ada Standard*.

For the purposes of this International Standard, the terms and definitions given in the Ada Standard and the following apply.

Additional terms are defined throughout this International Standard, indicated by *italic* type. Terms explicitly defined in this International Standard are not to be presumed to refer implicitly to similar terms defined elsewhere. Terms not defined in this International Standard and the Ada Standard are to be interpreted according to the *Webster's Third New International Dictionary of the English Language*. Informal descriptions of some terms are also given in Annex A, “Glossary”.

“ASIS” is used in reference to the acronym “Ada Semantic Interface Specification.” “Asis” is used in reference to the package Asis.

Section 2: ASIS technical concepts

2.1 Ada compilation environment

ASIS is an interface between an Ada environment as defined by the Ada Standard and any tool requiring information from this environment, as shown in Figure 2.

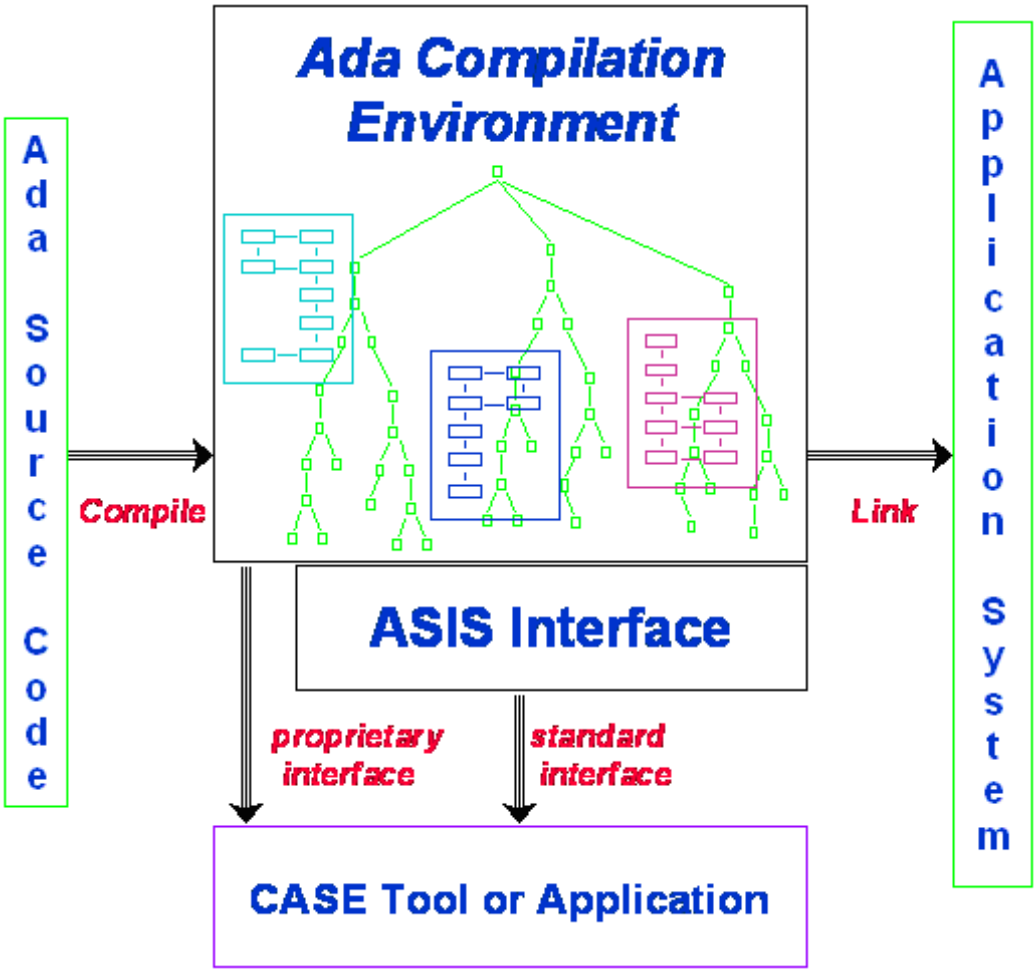

Figure 2 — ASIS as interface to Ada compilation environment

2.1.1 Ada environment

The Ada Standard, 10.1.4(1) provides a notion for this compilation *environment* as: “Each compilation unit submitted to the compiler is compiled in the context of an environment declarative_part (or simply environment), which is a conceptual declarative_part that forms the outermost declarative region of the context of any compilation. At run time, an environment forms the declarative_part of the body of the environment task of a partition.”

2.1.2 ASIS notion of the Ada compilation environment

The mechanisms for creating an environment and for adding and replacing compilation units within an environment are implementation-defined. In some implementations, environments are represented by a persistent database, while in others they are not. Consequently, ASIS requires the user of the interface (i.e., ASIS application) to establish the compilation environment. This is done through the context.

The *context* is a view of an Ada environment, and defines a set of compilation units and configuration pragmas to be processed by an ASIS application. ASIS provides any information from a context by treating this set as if its elements make up an environment declarative part. ASIS requires an application to identify the view of the environment to be provided by the context using the procedure Asis.Ada_Environments.Associate, as shown in Figure 3. ASIS may process several different contexts at a time.

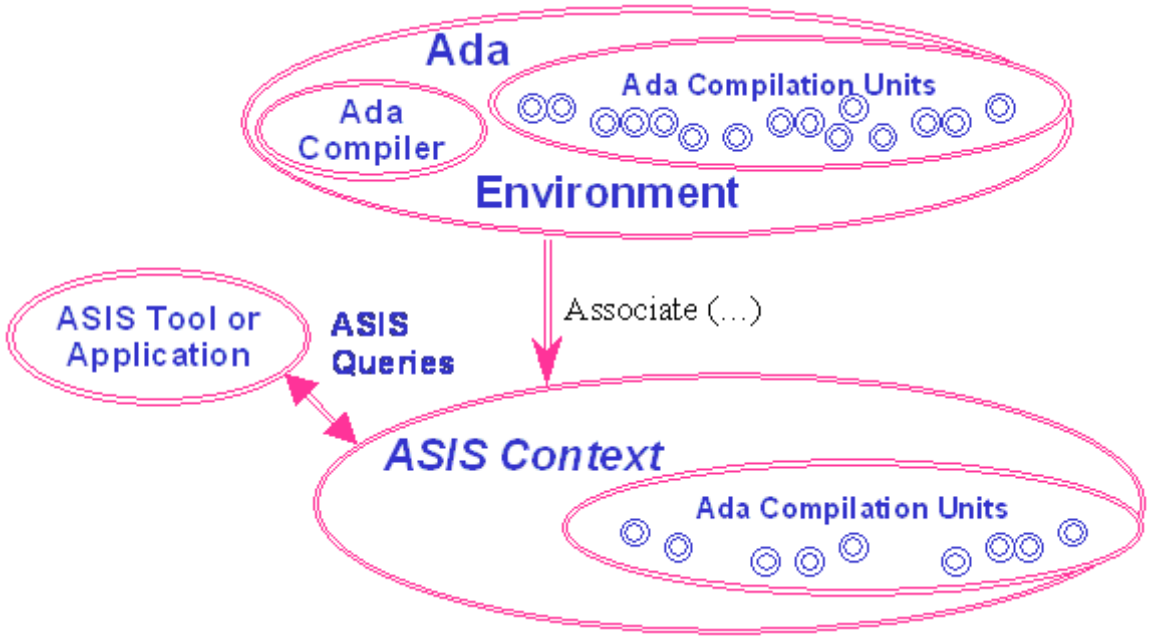

Figure 3 — Application interface to ASIS Context

A context may have one or more Compilation_Units. ASIS has defined Compilation_Unit as an ASIS private type. This type has values which denote an Ada compilation unit or configuration pragma from the environment. Compilation_Unit also is an abstraction, which represents information about some physical object from the “external world”. This physical object is treated by the underlying Ada implementation as the corresponding Ada compilation unit or as a result of compiling a configuration pragma. An ASIS *compilation unit* includes the notion of some implementation-defined way to associate the corresponding ASIS object with some physical external object. This is necessary to support ASIS queries such as Time_Of_Last_Update and Text_Name which have no relation to the Ada Standard-defined notion of an Ada compilation unit.

To facilitate the use of context, implementations may support the use of *containers* which are logical collections of ASIS compilation units. For example, some containers can hold compilation units that declare Ada predefined types; another container can hold implementation-defined packages. Containers provide an implementation-defined way of grouping the compilation units accessible for an ASIS application through the ASIS queries.

2.1.3 Illegal / inconsistent units in the compilation Environment

Ada Implementation permissions allow for illegal and inconsistent units to be in the environment. Because the contents of the Ada environment are Ada-implementation-defined, the ASIS context can contain illegal compilation units. The use of ASIS can result in the exception ASIS_Failed being raised if the Ada environment includes such units.

2.2 ASIS queries

ASIS queries are provided in the form of structural (syntactic) and semantic queries to the Ada compilation environment.

2.2.1 Structural queries

Structural queries are those ASIS queries which provide the top-down decomposition and reverse bottom-up composition of the compilation unit according to its syntax structure. Structural queries are the primary component of the syntactic subsystem, and are only found in that subsystem. These structural queries are further characterized as:

- "Black-box" queries are those ASIS queries which produce information about compilation units.
- "White-box" queries are those ASIS queries which produce information about lexical elements of compilation units.

2.2.2 Semantic queries

Semantic queries are those ASIS queries which express semantic properties (that is, the meaning) of constructs in the compilation unit.

Semantic queries in the syntactic subsystem express semantic properties of ASIS Elements in terms of other Elements. There are three kinds of semantic queries in the syntactic subsystem of ASIS:

- Semantic queries about Elements,
- Semantic queries about Compilation Units, and
- Semantic queries about Dependence Order.

In addition, semantic queries are provided by the semantic subsystem(see section 23) which defines a largely self-contained set of packages defining the semantics of the program in terms of Views and Declarations of various kinds of semantic entities, such as Objects and Subtypes. The semantic subsystem is linked to the structural queries through Element_Denoting_View, and from the structural queries through Corresponding_View, Corresponding_Subtype_View, and Corresponding_View_Declaration.

2.2.3 General ASIS query processing

Both structural (syntactic) and semantic queries are facilitated through the notion of ASIS elements and their kinds. Most ASIS queries (outside the semantic subsystem) provide for the processing of specific constructs and lists thereof with respect to the Ada Standard.

2.2.3.1 Elements and element kinds

The base object in ASIS is the Asis.Element. *Element* is a common abstraction used to represent syntactic constructs (both explicit and implicit) occurring within ASIS compilation units. Elements correspond to nodes of a hierarchical tree representation of an Ada program. Most elements of the tree have child elements. These children can appear as single elements (possibly with children themselves) or as a list of elements (also possibly with children). As an example, consider an Ada object declaration having three sub-parts or children:

- A list of identifiers,
- A reference to a subtype (subtype indication),
- An initialization expression (possibly absent)

Thus, the declaration:

```
A, B : Latitude := 0.0;
```

has a corresponding tree as in Figure 4.

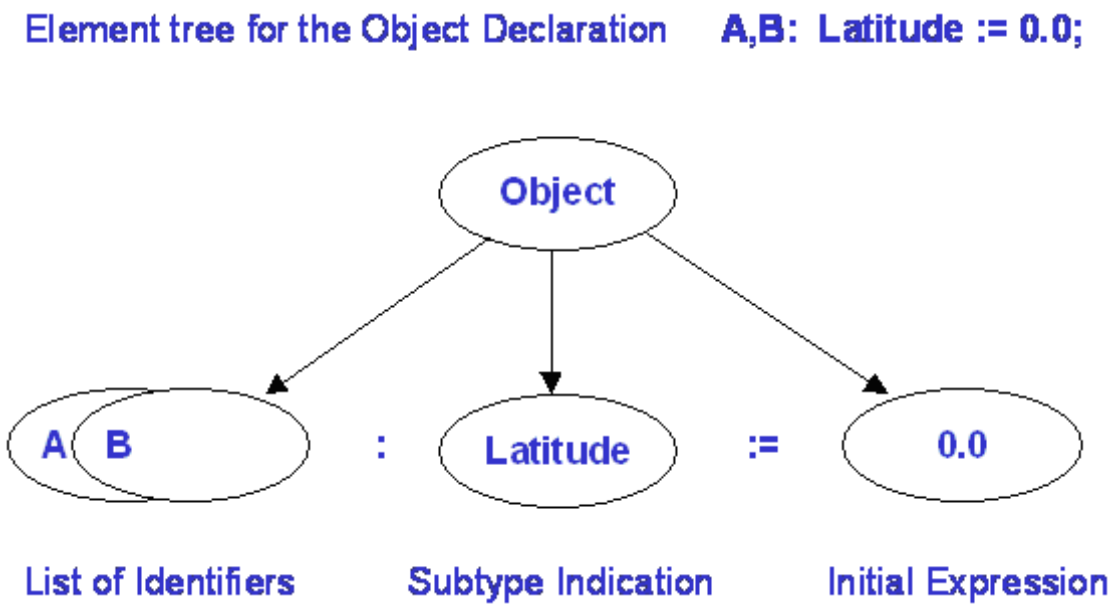

Figure 4 — Syntactic tree representation of an Ada object declaration

ASIS Elements are either *explicit elements*, representing a language construct that appears explicitly in the program text for the compilation unit, or *implicit elements*, representing a language construct that does not exist in the program text for the compilation unit, but could occur at a given place in the program text as a consequence of the semantics of another construct (e.g., an implicit declaration, a generic instantiation).

ASIS provides the ability to visit elements of the tree and ask questions of each element. One key hierarchy of questions begins with: “What kind of element do I have?’ Elements of the highest level of the ASIS hierarchy are classified into kinds. The following table identifies the high level hierarchy of kinds, the primary ASIS package to support processing of those kinds, and the references for those kinds in the Ada Standard.

A_Defining_Name	Asis.Definitions	Ada Standard 3, 6
A_Declaration	Asis.Declarations	Ada Standard 3, 5, 6, 7, 8, 9, 10, 11, 12
A_Definition	Asis.Definitions	Ada Standard 3, 7, 9, 12
An_Expression	Asis.Expressions	Ada Standard 2, 4
A_Statement	Asis.Statements	Ada Standard 5, 6, 9, 11, 13
A_Path	Asis.Statements	Ada Standard 5, 9
A_Clause	Asis.Clauses	Ada Standard 8, 10, 13
An_Association	Asis.Expressions	Ada Standard 2, 3, 4, 6, 12
An_Exception_Handler	Asis.Statements	Ada Standard 11.2
A_Pragma	Asis.Elements	Ada Standard 2, 10, 11, 13, B, G, H, I, L

The function Asis.Elements.Element_Kind classifies any element into one of these kinds. Once the client knows that an element is a declaration, for example, it can further classify the element as to what kind of declaration it is with the Asis.Elements.Declaration_Kind function. This leads to case structures like the following.

Examples

Case statement to classify elements:

```
case Asis.Elements.Element_Kind (My_Element) is              -- 13.6
   when Asis.A_Declaration =>                                -- 3.7.1
      case Asis.Elements.Declaration_Kind (My_Element) is    -- 13.9
         when Asis.A_Variable_Declaration =>                 -- 3.7.4
            { statement }
         when others =>
            null;
      end case;
   when Asis.A_Statement =>                                  -- 3.7.1
      case Asis.Elements.Statement_Kind (My_Element) is      -- 13.36
         when Asis.A_Block_Statement =>                      -- 3.7.22
            { statement }
         when others =>
            null;
      end case;
   when others =>
      null;
end case;
```

In this example, variable declarations and block statements will presumably be processed further. All other element kinds are ignored by falling into the null **when others** alternative.

2.2.3.2 Processing specific constructs

Once this level of classification is determined, ASIS provides a set of functions for processing a specific statement, declaration, or other element kinds. The following functions are available for processing object declarations:

```
function Names (Declaration : Asis.Declaration)                      -- 15.1
      return Asis.Defining_Name_List;
function Object_Declaration_Subtype (Declaration : Asis.Declaration)     -- 15.9
      return Asis.Definition;
function Initialization_Expression (Declaration : Asis.Declaration)   -- 15.10
      return Asis.Expression;
```

These functions correspond to each of the three parts of an object declaration. Once it is classified into its kind, each element can be processed further with the provided functions. Some functions “traverse” to other child elements such as the Object_Declaration_Subtype and Initialization_Expression functions above. Since the kind of element that is returned for each of these functions is already known, processing can continue directly with the functions in the Asis.Definitions package (Section 16) that accept A_Subtype_Indication elements or the Asis.Expressions package (Section 17) that accept An_Expression elements. The exception ASIS_Inappropriate_Element is raised whenever an ASIS function is passed an element it is not intended to process. Parameter names and subtype names in the function specifications indicate what kind of input element is expected and what kind of element is returned.

2.2.3.3 Element list processing

Lists of elements are processed with a loop iteration scheme in the following manner:

Examples

Loop iteration scheme

```
   List : constant Asis.Element_List :=                  -- 3.5
       <ASIS function returning a list>;
   An_Element : Asis.Element;                            -- 3.4
begin
   for I in List'Range loop
       An_Element := List (I);
       Process (An_Element);
   end loop;
```

Functions that return Element_List types generally indicate what type of elements are in the list they return. Thus, processing can sometimes continue with specific calls without first asking what type of element is being processed.

2.2.3.4 Operations that apply to all elements

Figure 5 depicts operations which, in general, apply to all elements.

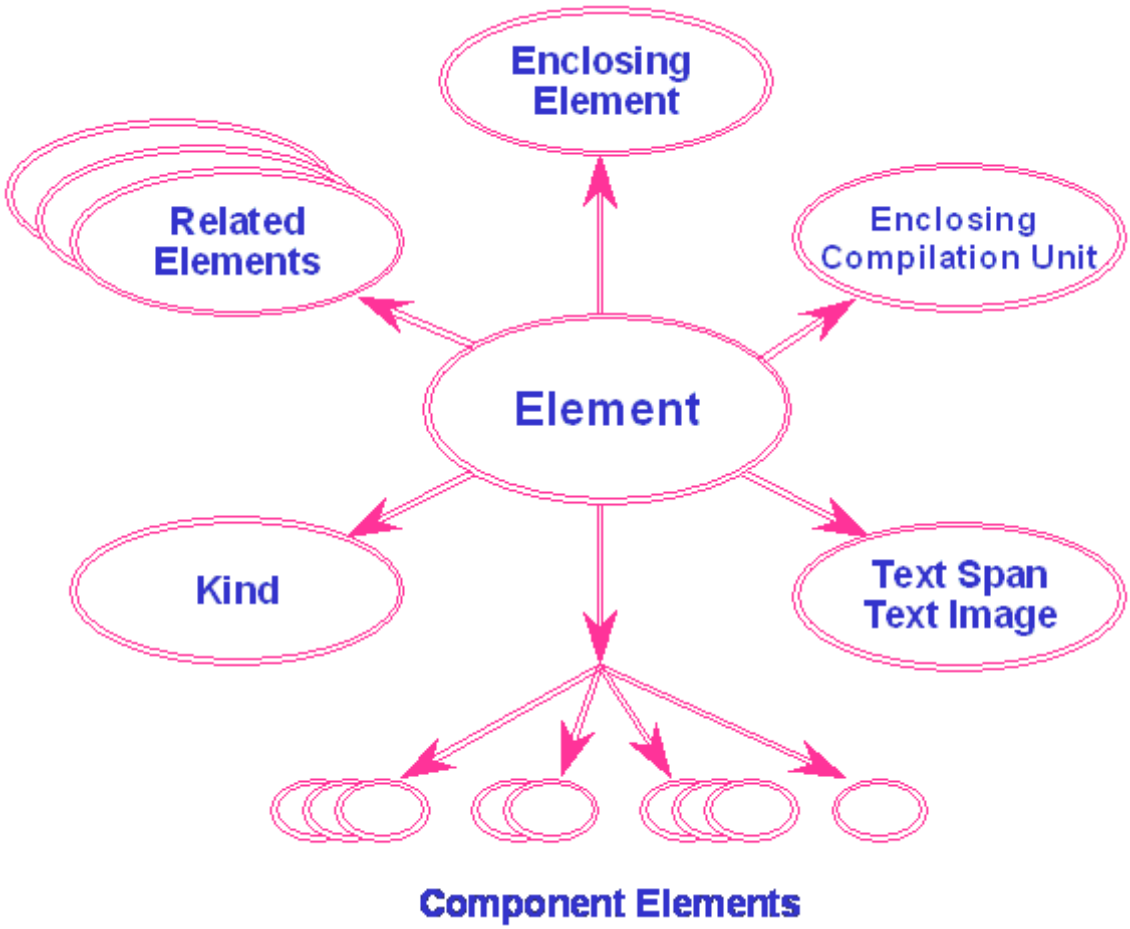

Figure 5 — Operations on elements

The ASIS packages provide some general interfaces that operate on all nodes, such as:

- Asis.Elements.Element_Kind (13.6) – Returns the Element_Kind for the element. Once the Element_Kind is known, the Element can be decomposed into its component elements.
- Asis.Elements.Enclosing_Element (13.50) – Returns the element that contains the current element (one element up in the tree hierarchy).
- Asis.Elements.Enclosing_Compilation_Unit (13.2) – Returns the Compilation_Unit that contains the given element.
- Asis.Text.First_Line_Number (20.6) – Returns the first line number in which the text of the element resides.
- Asis.Text.Last_Line_Number (20.7) – Returns the last line number in which the text of the element resides.
- Asis.Text.Element_Span (20.8) – Returns the span for the element.
- Asis.Text.Lines (20.17) – Returns a list of lines for the element.
- Asis.Text.Element_Image (20.21) – Returns the program text image of any element.

- Asis.Elements.Hash (13.56) – Returns a convenient name for an object of type Asis.Element.
- Asis.Ids.Create_Id (21.7) – Returns a unique Id value corresponding to this element.
- Asis.Elements.Is_Nil (13.40) – Determines whether an element is nil. Some functions return a Nil_Element when a potential element does not exist in the program. This is true for the Initialization_Expression function above when no initial value is present in the declaration.

2.2.3.5 Semantic references

References from one part of an Ada program to another can be traversed with functions whose name begins with “Corresponding_”, such as: Corresponding_Children, Corresponding_-Declaration, Corresponding_Body, Corresponding_Expression_Type, Corresponding_-Type_Declaration, Corresponding_Name_Definition, Corresponding_Name_Declaration, etc. If an element references another element, the user can traverse to the referenced element with such a function. A Nil_Element is returned if no definition traversal is possible.

The most fundamental such function is Corresponding_Name_Definition. This function traverses from a Name to the corresponding defining name, represented by an element of kind A_Defining_Name. From this point a traversal using the Enclosing_Element function can be used to arrive at the declaration of the corresponding program entity. Corresponding_-Name_Declaration combines these two traversals into a single operation. Corresponding_-Expression_Type traverses from an Expression to the declaration of its type.

For example, the user can traverse to the declaration of the type for an object named in an expression with the Corresponding_Expression_Type function as shown in Figure 6. Figure 6 also depicts links using the semantic query Corresponding_Name_Declaration. (Note: the thin lined-arrows depict syntactic queries while the thick-lined arrows depict semantic queries.)

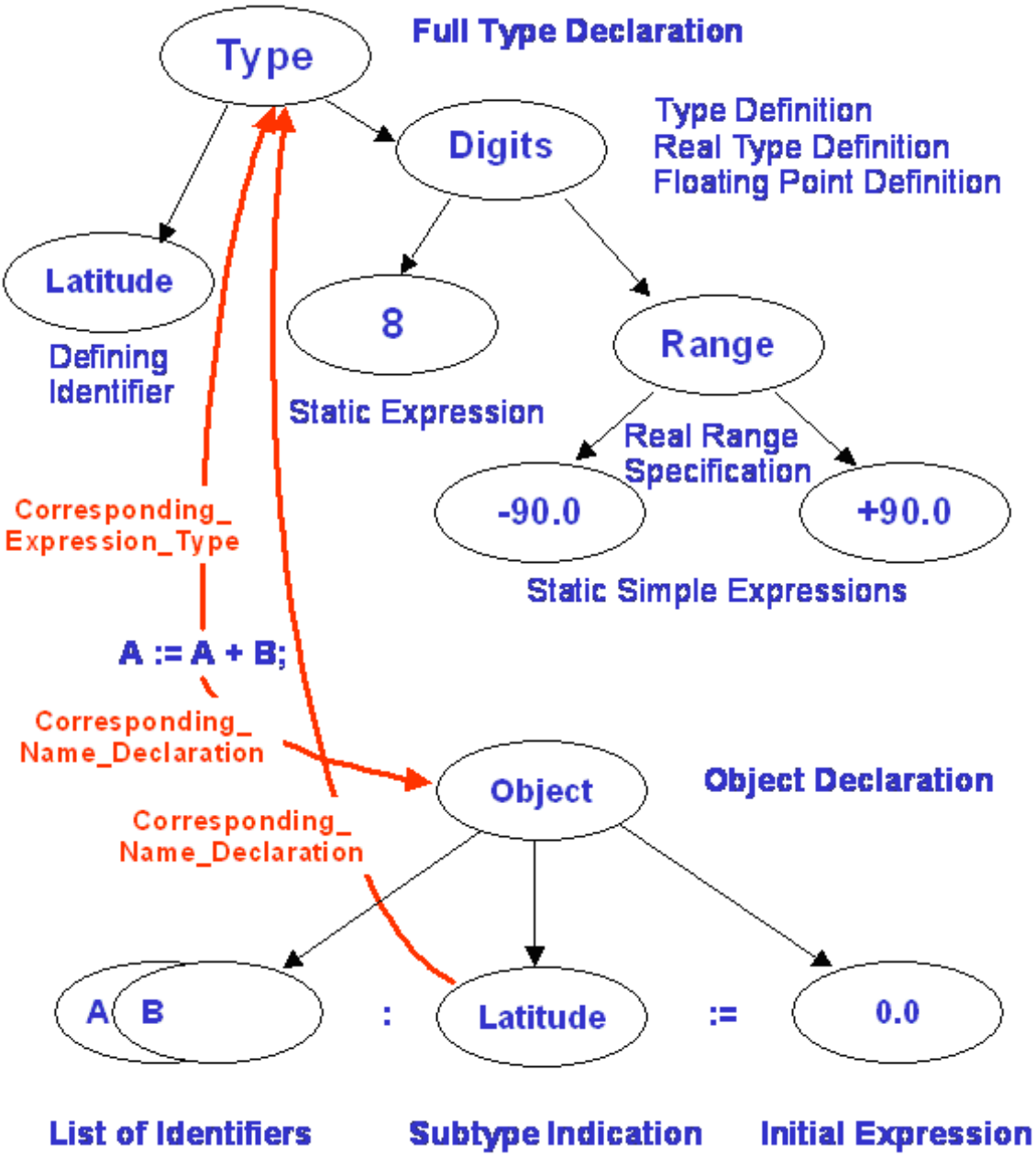

Figure 6 — Semantic reference using corresponding queries

2.3 ASIS package architecture

Figure 7 depicts the package architecture for the ASIS interface (not including the semantic subsystem). A tool or application using ASIS has visibility to the entire declarative region of package Asis including all child packages. To get a better understanding of how these packages are used, see the examples in Annex B.

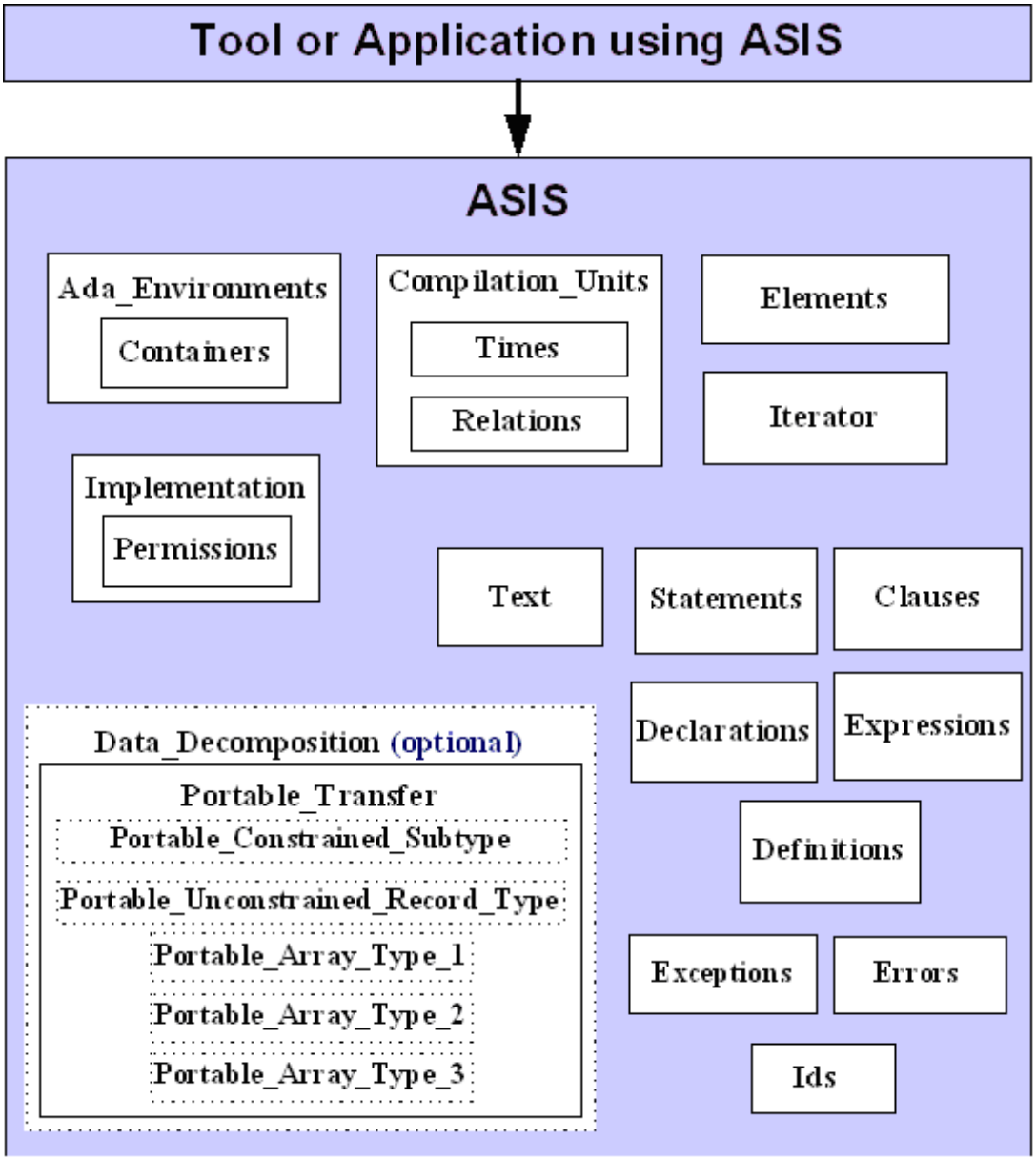

Figure 7 — ASIS package architecture

package Asis – Package Asis, together with its children, provides the interface between an Ada environment (as defined by the Ada Standard) and any tool requiring information from it. Valuable semantic and syntactic information is made available via queries through child packages of package ASIS.

Package Asis contains common types and subtypes used for the ASIS interface and its child packages. Important common types include (see Section 3):

- Type Context helps identify the compilation units considered to be analyzable as part of the Ada compilation environment.
- Type Element is an abstraction of constructs within a logical Ada syntax tree.
- Element Kinds are a set of enumeration types that characterize the constructs of Ada syntax.
- Type Compilation_Unit is an abstraction that represents an Ada compilation unit.
- Unit Kinds are a set of enumeration types describing the various kinds of compilation units.

- Type Traverse_Control provides a mechanism to control iterative traversals of a logical syntax tree.
- Type Program_Text provides an abstraction for the program text for a program's source code.

Package Asis also encapsulates implementor-specific declarations, which are made available to ASIS and its client applications in an implementor-independent manner. Package ASIS is the root of the ASIS interface. All other packages are child packages of package Asis. These packages are:

- **Asis.Ada_Environments** – This child package encapsulates a set of queries that map physical Ada compilation and program execution environments to logical ASIS environments. An ASIS Context is associated with some set of Ada compilation units maintained by an underlying Ada implementation. After this association has been made, this set of units is considered to be part of the compile-time Ada environment, which forms the outermost context of any compilation, as specified in section 10.1.4 of the Ada Standard. This same environment context provides the implicit outermost anonymous task during program execution. If an Ada implementation supports the notion of a program library or “library” as specified in section 10(2) of the Ada Standard, then an ASIS Context value can be mapped onto one or more implementor libraries represented by Containers. More than one context may be manipulated at a time. Important interfaces include: Associate, Dissociate, Open, and Close (see Section 8). The type Container and its supporting functions are provided in the child package **Asis.Ada_Environments.Containers** (see Section 9).
- **Asis.Implementation** – This child package provides operations to initialize and finalize the ASIS interface. It also provides queries for the error status of the ASIS implementation (see Section 6). Its child package **Asis.Implementation.Permissions** provides queries to determine options used by an implementation (see Section 7).
- **Asis.Compilation_Units** – This child package encapsulates a set of queries that implement the ASIS Compilation_Unit abstraction. It defines queries that deal with Compilation_Units and the gateway queries between Compilation_Units, Elements, and Ada_Environments. More than one compilation unit may be manipulated at one time (see Section 10). The child package **Asis.Compilation_Units.Times** encapsulates the time related functions used within ASIS (see Section 11). A second child package, **Asis.Compilation_Units.Relations** encapsulates the semantic relationship concepts used in ASIS. Relation queries provide references to compilation units that are related, in some specific fashion, to one or more given compilation units (see Section 12).
- **Asis.Iterator** – This child package encapsulates the Traverse_Element mechanism to perform an iterative traversal of a logical syntax tree. During the traversal, ASIS can analyze the various elements present and provide application processing via generic procedures instantiated by the application using queries to decompose ASIS elements into the logical semantic tree of the Ada program (see Section 14). This key mechanism is the heart of most ASIS tools; examples of its use are provided in Annex B.
- **Asis.Elements** – This child package encapsulates a set of queries that operate on all elements and some queries specific to A_Pragma elements. The element kinds, defined in package Asis, are defined as a set of enumeration types describing the various kinds of elements. ASIS offers a hierarchical classification of elements. At the highest level, the Element_Kinds type provides literals that define "kinds" or classes into which all non-nil elements are grouped. Element_Kinds are: Not_An_Element, A_Pragma, A_Defining_Name, A_Declaration, A_Definition, An_Expression, An_Association, A_Statement, A_Path, A_Clause, and An_Exception_Handler. Elements in each of the Element_Kinds classes, with the exception of An_Exception_Handler, can be further classified by a subordinate kind at the next level in the hierarchy (see Section 13).
- **Asis.Clauses** – This child package encapsulates a set of queries that operate on the A_Clause element (see Section 19).

- **Asis.Declarations** – This child package encapsulates a set of queries that operate on A_Defining_Name and A_Declaration elements (see Section 15).
- **Asis.Definitions** – This child package encapsulates a set of queries that operate on A_Definition elements (see Section 16).
- **Asis.Expressions** – This child package encapsulates a set of queries that operate on An_Expression and An_Association elements (see Section 17).
- **Asis.Statements** – This child package encapsulates a set of queries that operate on A_Statement, A_Path, and An_Exception_Handler elements (see Section 18).
- **Asis.Text** – This child package encapsulates a set of operations to access the text of ASIS Elements. This text is represented as logical *lines* from the source code of the external representation of a compilation unit. Type Line is defined to support program text operations (see Section 20). It assumes no knowledge of the existence, location, or form of the program text.
- **Asis.Errors** – This child package provides an enumeration type identifying the error kinds used for the ASIS interface (see Section 4).
- **Asis.Exceptions** – This child package identifies all defined ASIS exceptions (see Section 5).
- **Asis.Ids** – This child package encapsulates a set of operations and queries that implement the ASIS Id abstraction. An Id is a way to identify a particular element (i.e., a unique reference to an Element) which is efficient and persistent as long as the environment is not recompiled (see Section 21).
- **Asis.Data_Decomposition** – This optional child package encapsulates a set of operations to decompose data values using the ASIS type information and a portable data stream, representing a data value of that type (see Section 22).
- **Asis.Views** – This child package provides the basis of the semantic subsystem. A view represents a semantic entity in an Ada program, whether or not the entity has a syntactic representation. This package includes mechanisms for finding the associated element (if any). (see Section 23).
- **Asis.Declarations.Views** – This child package provides a path to move from a syntactic declaration (an element) to a semantic one (a view).
- **Asis.Definitions.Views** – This child package provides a path to move from a syntactic definition (an element) to a semantic one (a view).
- **Asis.Expressions.Views** – This child package provides a path to move from a syntactic expression (an element, which can be a value or object) to a semantic one (a view).
- **Asis.Program_Units** – This child package provides a semantic representation of program units, including ones that don't exist explicitly, like inherited subprograms and entities declared by generic instances. It also provides a set of queries on this semantic representation.
- **Asis.Subtype_Views** – This child package provides queries on semantic views of (all) subtypes.
- **Asis.Subtype_Views.Elementary** – This child package provides queries specifically for semantic views of elementary subtypes.
- **Asis.Subtype_Views.Composite** – This child package provides queries specifically for semantic views of composite subtypes.
- **Asis.Object_Views** – This child package provides queries on semantic views of values and objects.
- **Asis.Object_Views.Access_Views** – This child package provides queries on semantic views of values and objects of access types.

- **Asis.Profiles** – This child package provides the definition of and queries on subprogram profiles.
- **Asis.Callable_Views** – This child package provides queries on semantic views of entities that can be called (subprograms and entries).
- **Asis.Package_Views** – This child package provides queries on semantic views of packages units.
- **Asis.Generic_Views** – This child package provides queries on semantic views of generic units.
- **Asis.Exception_Views** – This child package provides queries on semantic views of exceptions.
- **Asis.Statement_Views** – This child package provides queries on semantic views of statements.

2.4 Application use

The ASIS Interface is provided through child packages of package ASIS. Complete executable examples of application use are provided in Annex B. This section discusses the establishment of ASIS context, required sequencing of calls, erroneous applications, and usage rules.

2.4.1 Establishing ASIS context

An application using ASIS has a context clause for package ASIS and the child packages needed by the application.

An application using all non-obsolescent child packages includes the following in its context clause:

```
with Asis, Asis.Errors,
  Asis.Compilation_Units, Asis.Compilation_Units.Times,
  Asis.Compilation_Units.Relations, Asis.Ada_Environments, Asis.Implementation,
  Asis.Exceptions, Asis.Elements, Asis.Iterator, Asis.Declarations,
  Asis.Expressions, Asis.Clauses, Asis.Definitions, Asis.Statements, Asis.Text,
  Asis.Ids, Asis.Data_Decomposition, Asis.Views,
  Asis.Program_Units, Asis.Subtype_Views, Asis.Subtype_Views.Elementary,
  Asis.Subtype_Views.Composite, Asis.Object_Views,
Asis.Object_Views.Access_Views,
  Asis.Profiles, Asis.Callable_Views, Asis.Package_Views,
  Asis.Generic_Views, Asis.Exception_Views, Asis.Statement_Views,
  Asis.Declarations.Views, Asis.Definitions.Views,
  Asis.Expressions.Views;
```

Only these packages should be referenced (**with**ed) by a newly written portable application. Obsolescent packages should not be used in new applications; other packages, which may be present in a specific ASIS implementation, are not part of this International Standard.

2.4.2 Required sequencing of calls

An ASIS application shall use the following required sequencing of calls:

```
a)  Asis.Implementation.Initialize;          -- Initialize the ASIS interface
b)  Asis.Ada_Environments.Associate(..);     -- Name an Ada Environment
c)  Asis.Ada_Environments.Open(..);          -- Access an Ada Environment
d)  Use the various ASIS queries.            -- Fetch a unit, its attributes,
                                             -- get its Unit_Declaration element,
                                             -- traverse its elements, etc.
e)  Asis.Ada_Environments.Close(..);         -- Drop access to an Ada Environment
f)  Asis.Ada_Environments.Dissociate(..);    -- Release the Ada Environment name
g)  Asis.Implementation.Finalize;            -- Release all resources
```

These calls may be used in a loop. More than one element may be manipulated at one time. (The exact number is subject to implementation/target specific limitations). An element, obtained

from a compilation unit, can continue to be manipulated while the Context, from which the element's compilation unit was obtained, remains open.

2.4.3 Notional ASIS application

ASIS Applications may take on many forms. This section is intended to present a notional ASIS Application using the example of a simple restrictions checker. A restrictions checker is intended to visit every element in an ASIS context to determine if a violation of a safety-critical check has been made. The ASIS context could include all compilation units in the Ada application. Such a restrictions checker might contain a large number of restrictions to check. The example restrictions checker looks for violations of two safety-critical guidelines:

a) Short circuit operators are always used (i.e., **or else** and **and then** are used and **or** and **and** are not used).
b) Tasks are declared at the library level.

A procedure, named Process_Element, is created which contains calls to procedures Check_Short_Circuit and Check_Library_Level_Task, which performs a restrictions check for each of our safety-critical guidelines above.

```
procedure Process_Element (Elem       : in Asis.Element;                -- 3.4
                           Control    : in out Asis.Traverse_Control;   -- 3.11
                           Dummy      : in out Boolean) is

begin

   Check_Short_Circuit (Elem);
   Check_Library_Level_Task (Elem);

   -- Additional guidelines can be checked here.

end Process_Element;
```

This procedure will process each element in the Context as controlled by an instantiation to Traverse_Element. The body of the Check_Short_Circuit and Check_Library_Level_Task identify the processing to be performed on each Element.

The Check_Short_Circuit procedure is passed the current Element to be evaluated. The Operator_Kinds function identifies the operator kind. If the Element happens to be An_And_Operator or An_Or_Operator, then a violation exists and must be reported by identifying the line number of the violation. Otherwise, there is no processing for this Element.

```
procedure Check_Short_Circuit (Elem : in Asis.Element) is              -- 3.4
   Op_Kind : Asis.Operator_Kinds :=                                    -- 3.7.20
                Asis.Elements.Operator_Kind (Elem);                    -- 13.33

begin
   case Op_Kind is

      when Asis.An_And_Operator =>                                     -- 3.7.20
         Put_Line ("Violation of Short Circuit Operator guideline:");
         Put ("-- Use of AND Operator at line ");
         Put (Asis.Text.Line_Number'Wide_Image                         -- 20.2
                  (Asis.Text.First_Line_Number (Elem)));               -- 20.6
         New_Line;

      when Asis.An_Or_Operator =>                                      -- 3.7.20
         Put_Line ("Violation of Short Circuit Operator guideline:");
         Put ("-- Use of OR Operator at line ");
         Put (Asis.Text.Line_Number'Wide_Image                         -- 20.2
                (Asis.Text.First_Line_Number (Elem)));                 -- 20.6
         New_Line;

      when others =>
         null;
   end case;

end Check_Short_Circuit;
```

The Check_Library_Level_Task checks to see if the Element parameter is a Declaration_Kind of A_Task_Type_Declaration, A_Protected_Type_Declaration, A_Single_Task_Declaration, or

A_Single_Protected_Declaration. If the Element is such a declaration, then a test Is_Library_Level is performed. If the task is not at the Library level, then a violation is reported along with its line number.

```
procedure Check_Library_Level_Task (Elem : Asis.Element) is            -- 3.4
begin
   case Asis.Elements.Declaration_Kind (Elem) is                       -- 13.9

      when Asis.A_Task_Type_Declaration |                              -- 3.7.4
           Asis.A_Protected_Type_Declaration |                         -- 3.7.4
           Asis.A_Single_Task_Declaration |                            -- 3.7.4
           Asis.A_Single_Protected_Declaration =>                      -- 3.7.4

         if not Is_Library_Level
            (Asis.Elements.Enclosing_Compilation_Unit(Elem)) then     -- 13.2
            Put_Line ("Violation of Tasking guideline:");
            Put ("-- Non-Library Level Task at Line:");
            Put (Asis.Text.Line_Number'Wide_Image                      -- 20.2
                (Asis.Text.First_Line_Number (Elem)));                 -- 20.6
            New_Line;
         end if;

       when others =>
          null;
    end case;
end Check_Library_Level_Task;
```

The function Is_Library_Level returns true when the Unit_Class of the Compilation_Unit is A_Public_Declaration.

```
function Is_Library_Level (CU : Asis.Compilation_Unit)              -- 3.8
   return Boolean is
begin

   case Asis.Compilation_Units.Unit_Class (CU) is                   -- 10.2
      when Asis.A_Public_Declaration =>                            -- 3.10.2
          return True;
      when others =>
          return False;
   end case;

end Is_Library_Level;
```

So far the procedure Process_Element has been created to check restrictions on an Element in a Compilation_Unit. The next step is to traverse all the Elements in a Compilation_Unit with this check. The following package, called Check_Compilation_Unit, does this with its procedure Find_Violations. Find_Violations prints the name of the Unit_Kind and name of each Compilation_Unit as it checks each element in the Compilation_Unit for restrictions using the procedure Check. Procedure Check is an instantiation of ASIS's Traverse_Element with Process_Element, containing the restriction checks. Traverse_Element provides a traversal of each element in a Compilation_Unit's logical syntax tree. The generic is instantiated with a state, a pre-operation, and a post-operation. In this example, Process_Element is the pre-operation which is executed when we land on each Element in the logical syntax tree. In this example, no processing is needed as we leave the Element, so the third generic parameter is the procedure No_Op. The state is not needed by the application. The package Check_Compilation_Unit provides the procedure Find_Violations for the selected Compilation_Unit. It traverses the logical syntax tree, finding and reporting the short circuit violations and task library level violations.

```
with Asis;
package Check_Compilation_Unit is

   procedure Find_Violations (CU : in Asis.Compilation_Unit);        -- 3.8

end Check_Compilation_Unit;

with Asis; with Asis.Elements; with Asis.Iterator; with Asis.Text;
with Ada.Wide_Text_Io; use Ada.Wide_Text_Io;
package body Check_Compilation_Unit is

   procedure Process_Element (Elem    : in Asis.Element;             -- 3.4
                              Control : in out Asis.Traverse_Control; -- 3.11
                              Dummy   : in out Boolean);
```

```
   procedure No_Op (Elem    : in Asis.Element;                          --  3.4
                    Control : in out Asis.Traverse_Control;             --  3.11
                    Dummy   : in out Boolean);

   procedure Check is new Asis.Iterator.Traverse_Element               --  14.1
                             (Boolean, Process_Element, No_Op);

   procedure Find_Violations (CU : Asis.Compilation_Unit) is            --  3.8
      Control : Asis.Traverse_Control := Asis.Continue;                --  3.11
      Dummy   : Boolean;
   begin
      Put_Line ("Processing " &
         Asis.Unit_Kinds'Wide_Image                                    --  3.10.1
            (Asis.Compilation_Units.Unit_Kind (CU))                    --  10.1
         & ": " &  (Asis.Compilation_Units.Unit_Full_Name (CU)));      --  10.19
      Check (Asis.Elements.Unit_Declaration (CU), Control, Dummy);
   end Check_Compilation_Unit;
```

The ASIS application is almost complete. A main subprogram is needed that contains the required sequencing of calls to initialize the ASIS interface, name the Ada environment, access the Ada environment, loop through all Compilation_Units in the ASIS Context with the Find_Violations procedure, and to close/release all ASIS resources. The Compilation_Units in the Context are placed into the Unit_List. This is achieved with the following main program, called My_Application.

```
with Asis;                                                             --  3
with Asis.Implementation;                                              --  6
with Asis.Ada_Environments;                                            --  8
with Asis.Compilation_Units;                                           --  10
with Check_Compilation_Unit;

procedure My_Application is
   My_Context : Asis.Context;                                          --  3.3

begin
   Asis.Implementation.Initialize;                                     --  6.6
   Asis.Ada_Environments.Associate (My_Context, "My Context");         --  8.3
   Asis.Ada_Environments.Open (My_Context);                            --  8.4

   declare
      Unit_List :  Asis.Compilation_Unit_List :=                       --  3.9
         Asis.Compilation_Units.Compilation_Units (My_Context);        --  10.10
   begin
      for I in Unit_List'Range loop
         case Asis.Compilation_Units.Unit_Origin (Unit_List (I)) is    --  10.3
            when Asis.An_Application_Unit =>                           --  3.10.3
               Check_Compilation_Unit.Find_Violations (Unit_List (I));
            when others => null;
         end case;
      end loop;
   end;

   Asis.Ada_Environments.Close (My_Context);                           --  8.5
   Asis.Ada_Environments.Dissociate (My_Context);                      --  8.6
   Asis.Implementation.Finalize;                                       --  6.8
end My_Application;
```

2.4.4 Erroneous applications

An ASIS application is *erroneous* if:

- It uses any ASIS query, other than those exported by Asis.Implementation, while Asis.Implementation.Is_Initialized = False.
- It attempts to use a Context before opening it (exceptions to this are Asis.Ada_Environments: Associate, Dissociate, and the Context query functions; these are the only ones that shall be used before opening a Context).
- It attempts to use a Context after closing it (exceptions to this are Asis.Ada_Environments: Associate, Dissociate, and the Context query functions; these are the only ones that shall be used after closing a Context).
- It attempts to Dissociate or Associate an open Context.

- It attempts to manipulate a Compilation_Unit whose Context has been closed.
- It attempts to obtain Compilation_Unit information from an unopened Context.
- It attempts to manipulate an Element from a Closed Context.

2.4.5 Usage rules

2.4.5.1 General usage rules

The following are general usage rules:

- All queries returning list values always return lists with a 'First of one (1).
- All queries with a Context parameter of mode IN raise ASIS_Inappropriate_Context if the Context value is not open. The Status is set to Value_Error.
- All queries with an Element or Line parameter attempt to detect the use of invalid values and raise ASIS_Inappropriate_Element in response. The Status is set to Value_Error. (An invalid value is one where the associated Context variable has been closed.) Not all ASIS implementations are able to detect the use of invalid values. An application that depends upon the success/failure of this invalid value detection is not portable.
- All queries other than simple Boolean or enumeration value queries, raise ASIS_Inappropriate_Element if passed an Element that is not appropriate to the query. The commentary for each query indicates the appropriate element kinds.
- It is generally inappropriate to mix elements of distinct subtypes (e.g., Passing a statement to a query expecting a declaration is inappropriate). It is also inappropriate to mix kinds of elements within a subtype when a query is expecting a specific kind. (i.e., Passing a type declaration to a query expecting a procedure declaration is inappropriate).
- Any query may raise ASIS_Failed with a Status of Obsolete_Reference_Error if the argument or result is itself, or is part of, an inconsistent compilation unit.
- Any Asis.Text query may raise ASIS_Failed with a Status of Text_Error if the program text cannot be located or retrieved for any reason such as renaming, deletion, corruption, or moving of the text or Ada environment.

2.4.5.2 Rules for processing queries for illegal/inconsistent context

The following provides general rules to identify processing when some inconsistent subset of Compilation Units is processed during semantic query processing. Processing of inconsistent unit sets is to a certain extent implementation-dependent; nevertheless, the following rules apply:

Semantic queries about elements:

Semantic queries across compilation boundaries may raise an exception if the units are inconsistent.

Semantic queries about compilation units:

- **Corresponding_Children**: If the declaration of a child is inconsistent with the argument of the query, neither declaration nor body is returned. If the declaration of a child is consistent with the argument, but the body is not, the declaration is returned, and for the body, the result of the Corresponding_Body query applied to the declaration is returned.
- **Corresponding_Parent_Declaration**: If a parent is inconsistent with a child passed as the argument, A_Nonexistent_Declaration shall be returned.
- **Corresponding_Declaration**: If the declaration of an argument Element is inconsistent with the argument, A_Nonexistent_Declaration shall be returned. (For a unit A_Procedure_Body or A_Function_Body kind the solution may be in any case to return Nil_Compilation_Unit if the unit is of A_Public_Declaration_And_Body kind).

- **Corresponding_Body**: If the argument Element requires a body to be presented to make up a complete partition containing this Element, but the Context does not contain the corresponding body, or the body contained in the Context is inconsistent with the argument Element, A_Nonexistent_Body shall be returned.
- **Subunits**: If a subunit is absent or if it is inconsistent with the argument Element, A_Nonexistent_Body shall be returned for it.
- **Corresponding_Subunit_Parent_Body**: If the corresponding body does not exist in the Context, or if it exists, but is inconsistent with the argument Element, then A_Nonexistent_Body shall be returned.

Semantic queries about semantic dependence order:

The Semantic_Dependence_Order query should never raise an exception when processing inconsistent unit (sub)sets. This query is the only means for an application to know if a given unit is consistent with (some of) its supporters (dependents), and therefore the related semantic processing can give valuable results for this unit.

2.4.5.3 Processing instantiations

Instantiations can always be analyzed in terms of the generic actual parameters supplied with the instantiation. A generic instance is a copy of the generic unit, and while there is no explicit (textual) specification in the program text, an implicit specification and body, if there is one, with the generic actual parameters is implied.

To analyze the implicit instance specification or body of a generic instantiation:

- Use Corresponding_Declaration to return the implicit expanded specification of an instantiation.
- Use Corresponding_Body to return the implicit body of an instantiation.
- Then analyze the specification or body with any appropriate queries.

To analyze the explicit generic specification or body referenced by a generic instantiation:

- Use Generic_Unit_Name to obtain the name of the generic unit.
- Then use Corresponding_Name_Declaration to get to the generic declaration.
- Then use Corresponding_Body to get to the body of the generic declaration.

Section 3: package Asis

The library package Asis shall exist. The package shall provide interfaces equivalent to those described in the following subclauses.

Package Asis encapsulates implementation-specific declarations, which are made available to ASIS and its client applications in an implementation-independent manner.

Package Asis is the root of the ASIS interface.

The Ada Semantic Interface Specification (ASIS) is an interface between an Ada environment as defined by the Ada Standard (ISO/IEC 8652:1995(E) and later documents) and any tool requiring information from this environment. An Ada environment includes valuable semantic and syntactic information. ASIS is an open and published callable interface which gives CASE tool and application developers access to this information. ASIS has been designed to be independent of underlying Ada environment implementations, thus supporting portability of software engineering tools while relieving tool developers from having to understand the complexities of an Ada environment's proprietary internal representation.

The following types are made visible directly through package Asis:

type ASIS_Integer
type ASIS_Natural
type ASIS_Positive
type List_Index
type Context
type Element
type Element_List
Element subtypes
Element Kinds (set of enumeration types)
type Compilation_Unit
type Compilation_Unit_List
Unit Kinds (set of enumeration types)
type Traverse_Control
subtype Program_Text

The ASIS interface uses string parameters for many procedure and function calls. Wide_String is used to convey ASIS environment information. Program_Text, a subtype of Wide_String, is used to convey program text. Throughout ASIS, Wide_String is to be interpreted according to the UTF-16 encoding defined in ISO/IEC 10646:2003, Annex Q. The Ada type String is not used in the ASIS interface. Neither the Ada types Character nor Wide_Character are used in the ASIS interface.

3.1 subtypes ASIS_Integer, ASIS_Natural, and ASIS_Positive

```
subtype ASIS_Integer is Implementation_Defined_Integer_Type;

subtype ASIS_Natural is ASIS_Integer range 0 .. ASIS_Integer'Last;
subtype ASIS_Positive is ASIS_Integer range 1 .. ASIS_Integer'Last;
```

Numeric subtypes that allow each ASIS implementation to place constraints on the lower and upper bounds of values. Whenever possible, the range of subtype ASIS_Integer should meet or exceed -(2**31-1) .. 2**31-1.

3.2 type List_Index

```
List_Index_Implementation_Upper : constant ASIS_Positive :=
    Implementation_Defined_Integer_Constant;
subtype List_Index is ASIS_Positive
    range 1 .. List_Index_Implementation_Upper;
```

List_Index is a numeric subtype used to establish the upper bound for list size.

3.3 type Context

The ASIS *Context* is a view of a particular implementation of an Ada environment. ASIS requires an application to identify that view of the Ada environment. An ASIS Context identifies an Ada environment as defined by the Ada Standard. The Ada environment is well defined for Ada implementations. the Ada Standard provides for an implementation-defined method to enter compilation units into the Ada environment. Implementation permissions allow for illegal and inconsistent units to be in the environment. The use of ASIS may result in the exception ASIS_Failed being raised if the Ada environment includes such units.

Defined by the implementation, an ASIS context is a way to identify a set of Compilation Units to be processed by an ASIS application. This may include things such as the pathname, search rules, etc., which are attributes of the Ada environment and consequently become part of the ASIS Context only because the context is a "view" of the Ada environment.

An ASIS Context is a handle to a set of compilation units accessible by an ASIS application. Because the contents of the Ada environment are (Ada-)implementation defined, the set of compilation units available from an ASIS context may be inconsistent, and may contain illegal compilation units. The contents are selected from the Ada environment in an implementation-defined manner. ASIS should allow multiple open contexts.

In the Context abstraction, a logical handle is associated with Name and Parameters values that are used by the implementation to identify and connect to the information in the Ada environment.

An ASIS Context is associated with some set of Ada compilation units maintained by an underlying Ada implementation or a stand-alone ASIS implementation. After this association has been made, this set of units is considered to be part of the compile-time Ada environment, which forms the outermost context of any compilation, as specified in section 10.1.4 of the Ada Standard. This same environment context provides the implicit outermost anonymous task during program execution.

Some implementations might not need explicit Name and/or Parameters values to identify their Ada environment. Other implementations might choose to implement the Ada environment as a single external file in which case the name and parameters values might simply supply the Name, Form, and any other values needed to open such a file. Context shall be an undiscriminated limited private type.

```
type Context is limited private;
Nil_Context : constant Context;

function "=" (Left  : in Context;
              Right : in Context)
              return Boolean is abstract;
```

Implementation Requirements

The concrete mechanism of this association is implementation-specific:

Each ASIS implementation provides the means to construct an ASIS Context value that defines the environment declarative_part or "context" from which ASIS can obtain library units.

A default-initialized context object is unassociated and not open (see 8.4). Nil_Context represents no context, and is unassociated and not open.

3.4 type Element

The *Element* type is a distinct type used to represent handles on the syntactic constructs that form the text of compilation units. Elements deal with the internal and then "textual" view of compilation units.

Operations are provided that retrieve one Element and two Element_Lists from an ASIS Compilation_Unit object:

a) A context clause represented by an Element_List containing with clauses, use clauses, and pragmas.

b) An Element associated with the declaration.

c) A list of pragmas, that are not part of the context clause but which nonetheless affect the compilation of the unit.

ASIS Elements are representations of the syntactic and semantic information available from most Ada environments.

The ASIS Element type shall be an undiscriminated private type.

```
type Element is private;
Nil_Element : constant Element;

function "=" (Left  : in Element;
              Right : in Element)
              return Boolean is abstract;
```

3.5 type Element_List

```
type Element_List is array (List_Index range <>) of Element;

Nil_Element_List : constant Element_List;
```

Type Element_List represents a list of elements.

3.6 subtypes of Element and Element_List

```
subtype Access_Type_Definition          is Element;
subtype Aspect_Clause                   is Element;
subtype Association                     is Element;
subtype Case_Statement_Alternative      is Element;
subtype Clause                          is Element;
subtype Component_Clause                is Element;
subtype Component_Declaration           is Element;
subtype Component_Definition            is Element;
subtype Constraint                      is Element;
subtype Context_Clause                  is Element;
subtype Declaration                     is Element;
subtype Defining_Name                   is Element;
subtype Definition                      is Element;
subtype Discrete_Range                  is Element;
subtype Discrete_Subtype_Definition     is Element;
subtype Discriminant_Association        is Element;
subtype Exception_Handler               is Element;
subtype Expression                      is Element;
subtype Formal_Type_Definition          is Element;
subtype Generic_Formal_Parameter        is Element;
subtype Identifier                      is Element;
subtype Name                            is Element;
subtype Parameter_Specification         is Element;
subtype Path                            is Element;
subtype Pragma_Element                  is Element;
subtype Range_Constraint                is Element;
subtype Record_Component                is Element;
subtype Record_Definition               is Element;
subtype Root_Type_Definition            is Element;
subtype Select_Alternative              is Element;
subtype Statement                       is Element;
subtype Subtype_Indication              is Element;
subtype Subtype_Mark                    is Element;
subtype Type_Definition                 is Element;
subtype Variant                         is Element;
```

```
subtype Aspect_Clause_List              is Element_List;
subtype Association_List                is Element_List;
subtype Component_Clause_List           is Element_List;
subtype Context_Clause_List             is Element_List;
subtype Declaration_List                is Element_List;
subtype Declarative_Item_List           is Element_List;
subtype Defining_Name_List              is Element_List;
subtype Definition_List                 is Element_List;
subtype Discrete_Range_List             is Element_List;
subtype Discriminant_Association_List   is Element_List;
subtype Discriminant_Specification_List is Element_List;
subtype Exception_Handler_List          is Element_List;
subtype Expression_List                 is Element_List;
subtype Generic_Formal_Parameter_List   is Element_List;
subtype Identifier_List                 is Element_List;
subtype Name_List                       is Element_List;
subtype Parameter_Specification_List    is Element_List;
subtype Path_List                       is Element_List;
subtype Pragma_Element_List             is Element_List;
subtype Record_Component_List           is Element_List;
subtype Statement_List                  is Element_List;
subtype Variant_Component_List          is Element_List;
subtype Variant_List                    is Element_List;
```

3.7 Element Kinds

Element Kinds are enumeration types describing various kinds of elements. These element kinds are used within package Asis.Elements.

3.7.1 type Element_Kinds

ASIS offers hierarchical classification of elements. At the highest level, the Element_Kinds type provides literals that define "kinds" or classes listed below into which all non-nil elements are grouped. Elements in each of the Element_Kinds classes, with the exception of An_Exception_Handler, can be further classified by a subordinate kind at the next level in the hierarchy. Several subordinate kinds have further sub-subordinate kinds.

For example, Element_Kinds'A_Declaration might be classified into Declaration_Kinds'A_Parameter_Specification which might be further classified into Mode_Kinds'An_In_Mode. This fully identifies the syntax of an element such as:

```
(Who : in Person)
```

All Element_Kinds and subordinate kinds Queries are in Asis.Elements.

It is not necessary to strictly follow the hierarchy; any element can be classified by any subordinate kind from any level. However, meaningful results will only be obtained from subordinate kinds that are appropriate. These are designated within the hierarchy shown below:

```
Element_Kinds          -> Subordinate Kinds
    (Read "->" as "is further classified by its")

A_Pragma               -> Pragma_Kinds

A_Defining_Name        -> Defining_Name_Kinds
                                -> Operator_Kinds

A_Declaration          -> Declaration_Kinds
                                -> Declaration_Origins
                                -> Mode_Kinds
                                -> Subprogram_Default_Kinds

A_Definition           -> Definition_Kinds
                                -> Type_Kinds
                                -> Formal_Type_Kinds
                                -> Access_Type_Kinds
                                -> Root_Type_Kinds
                                -> Constraint_Kinds
                                -> Discrete_Range_Kinds
```

```
   An_Expression         -> Expression_Kinds
                                    -> Operator_Kinds
                                    -> Attribute_Kinds
   An_Association        -> Association_Kinds
   A_Statement           -> Statement_Kinds
   A_Path                -> Path_Kinds
   A_Clause              -> Clause_Kinds
                                    -> Aspect_Clause_Kinds
   An_Exception_Handler
```

Type Element_Kinds provides general element classifications.

```
type Element_Kinds is (
   Not_An_Element,            -- Nil_Element
   A_Pragma,                  -- Asis.Elements
   A_Defining_Name,           -- Asis.Declarations
   A_Declaration,             -- Asis.Declarations
   A_Definition,              -- Asis.Definitions
   An_Expression,             -- Asis.Expressions
   An_Association,            -- Asis.Expressions
   A_Statement,               -- Asis.Statements
   A_Path,                    -- Asis.Statements
   A_Clause,                  -- Asis.Clauses
   An_Exception_Handler);     -- Asis.Statements
```

The comments list the ASIS package with queries for each kind.

3.7.2 type Pragma_Kinds

Type Pragma_Kinds provides classifications for pragmas.

```
type Pragma_Kinds is (
   Not_A_Pragma,                        -- An unexpected element
   An_All_Calls_Remote_Pragma,          -- E.2.3(5)
   An_Assert_Pragma ,                   -- 11.4.2 (3)
   An_Assertion_Policy_Pragma,          -- 11.4.2 (6)
   An_Asynchronous_Pragma,              -- E.4.1(3)
   An_Atomic_Pragma,                    -- C.6(3)
   An_Atomic_Components_Pragma,         -- C.6(5)
   An_Attach_Handler_Pragma,            -- C.3.1(4)
   A_Controlled_Pragma,                 -- 13.11.3(3)
   A_Convention_Pragma,                 -- B.1(7), M.1(5)
   A_Discard_Names_Pragma,              -- C.5(3)
   A_Detect_Blocking_Pragma,            -- D.13 (4)

   An_Elaborate_Pragma,                 -- 10.2.1(20)
   An_Elaborate_All_Pragma,             -- 10.2.1(21)
   An_Elaborate_Body_Pragma,            -- 10.2.1(22)
   An_Export_Pragma,                    -- B.1(5), M.1(5)
   An_Import_Pragma,                    -- B.1(6), M.1(5)
   An_Inline_Pragma,                    -- 6.3.2(3)
   An_Inspection_Point_Pragma,          -- H.3.2(3)
   An_Interrupt_Handler_Pragma,         -- C.3.1(2)
   An_Interrupt_Priority_Pragma,        -- D.1(5)
   A_Linker_Options_Pragma,             -- B.1(8)
   A_List_Pragma,                       -- 2.8(21)
   A_Locking_Policy_Pragma,             -- D.3(3)

   A_No_Return_Pragma,                  -- 6.5.1 (3)
   A_Normalize_Scalars_Pragma,          -- H.1(3)
   An_Optimize_Pragma,                  -- 2.8(23)
   A_Pack_Pragma,                       -- 13.2(3)
   A_Page_Pragma,                       -- 2.8(22)
   A_Partition_Elaboration_Policy_Pragma,    -- H.6 (3)
   A_Relative_Deadline_Pragma,          -- D.2.6 (2.2)
   A_Preelaborable_Initialization_Pragma,    -- 7.6 (5)
   A_Preelaborate_Pragma,               -- 10.2.1(3)
   A_Priority_Pragma,                   -- D.1(3)
   A_Priority_Specific_Dispatching_Pragma,   -- D.2.2 (2.2)
   A_Profile_Pragma,                    -- D.13 (2)
   A_Pure_Pragma,                       -- 10.2.1(14)
```

```
   A_Queuing_Policy_Pragma,           -- D.4(3)
   A_Relative_Deadline_Pragma,        -- D.2.6 (2.2)
   A_Remote_Call_Interface_Pragma,    -- E.2.3(3)
   A_Remote_Types_Pragma,             -- E.2.2(3)
   A_Restrictions_Pragma,             -- 13.12(3)
   A_Reviewable_Pragma,               -- H.3.1(3)
   A_Shared_Passive_Pragma,           -- E.2.1(3)
   A_Storage_Size_Pragma,             -- 13.3(63)
   A_Suppress_Pragma,                 -- 11.5(4)
   A_Task_Dispatching_Policy_Pragma,  -- D.2.2(2)
   An_Unchecked_Union_Pragma,         -- B.3.3 (3)
   An_Unsuppress_Pragma,              -- 11.5 (4.1)
   A_Volatile_Pragma,                 -- C.6(4)
   A_Volatile_Components_Pragma,      -- C.6(6)

   An_Implementation_Defined_Pragma,  -- 2.8(14)

   An_Unknown_Pragma);                -- Unknown to ASIS
```

The comments list a reference to the definition in the Ada Standard for each pragma.

3.7.3 type Defining_Name_Kinds

Type Defining_Name_Kinds classifies names defined by declarations and specifications.

```
type Defining_Name_Kinds is (

   Not_A_Defining_Name,               -- An unexpected element

   A_Defining_Identifier,             -- 3.1(4)
   A_Defining_Character_Literal,      -- 3.5.1(4)
   A_Defining_Enumeration_Literal,    -- 3.5.1(3)
   A_Defining_Operator_Symbol,        -- 6.1(9)
   A_Defining_Expanded_Name);         -- 6.1(7)
                                      -- program unit name defining_identifier
```

The comments list a reference to the definition in the Ada Standard for each defining name.

3.7.4 type Declaration_Kinds

Type Declaration_Kinds classifies declarations and specifications having defining names.

```
type Declaration_Kinds is (

   Not_A_Declaration,                       -- An unexpected element

   An_Ordinary_Type_Declaration,            -- 3.2.1(3)
      -- a full_type_declaration of the form:
      -- type defining_identifier [known_discriminant_part] is type_definition;

   A_Task_Type_Declaration,                 -- 9.1(2)
   A_Protected_Type_Declaration,            -- 9.4(2)
   An_Incomplete_Type_Declaration,          -- 3.2.1(2), 3.10(2)
   A_Private_Type_Declaration,              -- 3.2.1(2), 7.3(2)
   A_Private_Extension_Declaration,         -- 3.2.1(2), 7.3(3)

   A_Subtype_Declaration,                   -- 3.2.2(2)

   A_Variable_Declaration,                  -- 3.3.1(2)
   A_Constant_Declaration,                  -- 3.3.1(4)
   A_Deferred_Constant_Declaration,         -- 3.3.1(6), 7.4(2)
   A_Single_Task_Declaration,               -- 3.3.1(2), 9.1(3)
   A_Single_Protected_Declaration,          -- 3.3.1(2), 9.4(2)

   An_Integer_Number_Declaration,           -- 3.3.2(2)
   A_Real_Number_Declaration,               -- 3.5.6(2)

   An_Enumeration_Literal_Specification,    -- 3.5.1(3)

   A_Discriminant_Specification,            -- 3.7(5)
   A_Component_Declaration,                 -- 3.8(6)
   A_Return_Object_Specification            -- 6.5(2)

   A_Loop_Parameter_Specification,          -- 5.5(4)

   A_Procedure_Declaration,                 -- 6.1(4)
   A_Function_Declaration,                  -- 6.1(4)
```

```
   A_Parameter_Specification,                -- 6.1(15)
                                             --      -> Mode_Kinds
   A_Procedure_Body_Declaration,             -- 6.3(2)
   A_Function_Body_Declaration,              -- 6.3(2)

   A_Package_Declaration,                    -- 7.1(2)
   A_Package_Body_Declaration,               -- 7.2(2)

   An_Object_Renaming_Declaration,           -- 8.5.1(2)
   An_Exception_Renaming_Declaration,        -- 8.5.2(2)
   A_Package_Renaming_Declaration,           -- 8.5.3(2)
   A_Procedure_Renaming_Declaration,         -- 8.5.4(2)
   A_Function_Renaming_Declaration,          -- 8.5.4(2)
   A_Generic_Package_Renaming_Declaration,   -- 8.5.5(2)
   A_Generic_Procedure_Renaming_Declaration,-- 8.5.5(2)
   A_Generic_Function_Renaming_Declaration, -- 8.5.5(2)

   A_Task_Body_Declaration,                  -- 9.1(6)
   A_Protected_Body_Declaration,             -- 9.4(7)
   An_Entry_Declaration,                     -- 9.5.2(2)
   An_Entry_Body_Declaration,                -- 9.5.2(5)
   An_Entry_Index_Specification,             -- 9.5.2(2)

   A_Procedure_Body_Stub,                    -- 10.1.3(3)
   A_Function_Body_Stub,                     -- 10.1.3(3)
   A_Package_Body_Stub,                      -- 10.1.3(4)
   A_Task_Body_Stub,                         -- 10.1.3(5)
   A_Protected_Body_Stub,                    -- 10.1.3(6)

   An_Exception_Declaration,                 -- 11.1(2)
   A_Choice_Parameter_Specification,         -- 11.2(4)

   A_Generic_Procedure_Declaration,          -- 12.1(2)
   A_Generic_Function_Declaration,           -- 12.1(2)
   A_Generic_Package_Declaration,            -- 12.1(2)

   A_Package_Instantiation,                  -- 12.3(2)
   A_Procedure_Instantiation,                -- 12.3(2)
   A_Function_Instantiation,                 -- 12.3(2)

   A_Formal_Object_Declaration,              -- 12.4(2)  -> Mode_Kinds
   A_Formal_Type_Declaration,                -- 12.5(2)
   A_Formal_Procedure_Declaration,           -- 12.6(2)  -> Subprogram_Default_Kinds
   A_Formal_Function_Declaration,            -- 12.6(2)  -> Subprogram_Default_Kinds
   A_Formal_Package_Declaration,             -- 12.7(2)
   A_Formal_Package_Declaration_With_Box);   -- 12.7(3)
```

The comments list a reference to the definition in the Ada Standard for each kind of declaration; the subordinate kind (if any) is given as well.

The following Declaration_Kinds subtypes are not used by ASIS but are provided for the convenience of the ASIS user:

```
subtype A_Type_Declaration is Declaration_Kinds range
           An_Ordinary_Type_Declaration .. A_Private_Extension_Declaration;

subtype A_Full_Type_Declaration is Declaration_Kinds range
           An_Ordinary_Type_Declaration .. A_Protected_Type_Declaration;

subtype An_Object_Declaration is Declaration_Kinds range
           A_Variable_Declaration .. A_Single_Protected_Declaration;

subtype A_Number_Declaration is Declaration_Kinds range
           An_Integer_Number_Declaration .. A_Real_Number_Declaration;

subtype A_Renaming_Declaration is Declaration_Kinds range
           An_Object_Renaming_Declaration ..
           A_Generic_Function_Renaming_Declaration;

subtype A_Body_Stub is Declaration_Kinds range
           A_Procedure_Body_Stub .. A_Protected_Body_Stub;

subtype A_Generic_Declaration is Declaration_Kinds range
           A_Generic_Procedure_Declaration .. A_Generic_Package_Declaration;

subtype A_Generic_Instantiation is Declaration_Kinds range
           A_Package_Instantiation .. A_Function_Instantiation;

subtype A_Formal_Declaration is Declaration_Kinds range
           A_Formal_Object_Declaration ..
           A_Formal_Package_Declaration_With_Box;
```

3.7.5 type Overriding_Indicator_Kinds

Type Overriding_Indicator_Kinds classifies declarations and specifications having an overrriding indicator.

```
type Overriding_Indicator_Kinds is (
  Not_An_Overriding_Indicator,
  No_Overriding_Indicator,          -- 8.3.1 (2)
  An_Indicator_of_Overriding,       -- 8.3.1 (2)
  An_Indicator_of_Not_Overriding);  -- 8.3.1 (2)
```

3.7.6 type Declaration_Origins

```
type Declaration_Origins is (
   Not_A_Declaration_Origin,            -- An unexpected element
   An_Explicit_Declaration,             -- 3.1(5) explicitly declared in
                                        -- the text of a program, or within
                                        -- an expanded generic template
   An_Implicit_Predefined_Declaration,  -- 3.1(5), 3.2.3(1), A.1(2)
   An_Implicit_Inherited_Declaration);  -- 3.1(5), 3.4(6-35)
```

The comments list a reference to the definition in the Ada Standard for each kind of declaration origin.

3.7.7 type Mode_Kinds

```
type Mode_Kinds is (          -- 6.1 in 8652:1995
   Not_A_Mode,                -- An unexpected element
   A_Default_In_Mode,         -- procedure A(B : C);
   An_In_Mode,                -- procedure A(B : in C);
   An_Out_Mode,               -- procedure A(B : out C);
   An_In_Out_Mode);           -- procedure A(B : in out C);
```

3.7.8 type Subprogram_Default_Kinds

```
type Subprogram_Default_Kinds is (    -- 12.6 in 8652:1995
   Not_A_Default,             -- An unexpected element
   A_Name_Default,            -- with subprogram_specification is default_name;
   A_Box_Default,             -- with subprogram_specification is <>;
   A_Null_Default,            -- with subprogram_specification is null;
   A_Nil_Default);            -- with subprogram_specification;
```

Note that an abstract formal subprogram is indicated with the Has_Abstract query (see 13.10).

3.7.9 type Definition_Kinds

```
type Definition_Kinds is (
   Not_A_Definition,                -- An unexpected element
   A_Type_Definition,               -- 3.2.1(4)   -> Type_Kinds
   A_Subtype_Indication,            -- 3.2.2(3)
   A_Constraint,                    -- 3.2.2(5)   -> Constraint_Kinds
   A_Component_Definition,          -- 3.6(7)
   A_Discrete_Subtype_Definition,   -- 3.6(6)     -> Discrete_Range_Kinds
   A_Discrete_Range,                -- 3.6.1(3)   -> Discrete_Range_Kinds
   An_Unknown_Discriminant_Part,    -- 3.7(3)
   A_Known_Discriminant_Part,       -- 3.7(2)
   A_Record_Definition,             -- 3.8(3)
   A_Null_Record_Definition,        -- 3.8(3)
```

```
   A_Null_Component,                  -- 3.8(4)
   A_Variant_Part,                    -- 3.8.1(2)
   A_Variant,                         -- 3.8.1(3)

   An_Others_Choice,                  -- 3.8.1(5), 4.3.1(5), 4.3.3(5), 11.2(5)
   An_Access_Definition,              -- 3.10(6)
   An_Incomplete_Type_Definition,     -- 3.10.1(1)
   A_Tagged_Incomplete_Type_Definition, -- 3.10.1(2)

   A_Private_Type_Definition,         -- 7.3(2)
   A_Tagged_Private_Type_Definition,  -- 7.3(2)
   A_Private_Extension_Definition,    -- 7.3(3)

   A_Task_Definition,                 -- 9.1(4)
   A_Protected_Definition,            -- 9.4(4)

   A_Formal_Type_Definition);         -- 12.5(3)   -> Formal_Type_Kinds
```

The comments list a reference to the definition in the Ada Standard for each kind of definition; the subordinate kind (if any) is given as well.

3.7.10 type Type_Kinds

```
type Type_Kinds is (

   Not_A_Type_Definition,                  -- An unexpected element

   A_Derived_Type_Definition,              -- 3.4(2)
   A_Derived_Record_Extension_Definition,  -- 3.4(2)

   An_Enumeration_Type_Definition,         -- 3.5.1(2)

   A_Signed_Integer_Type_Definition,       -- 3.5.4(3)
   A_Modular_Type_Definition,              -- 3.5.4(4)

   A_Root_Type_Definition,                 -- 3.5.4(14), 3.5.6(3)
                                           --      -> Root_Type_Kinds
   A_Floating_Point_Definition,            -- 3.5.7(2)

   An_Ordinary_Fixed_Point_Definition,     -- 3.5.9(3)
   A_Decimal_Fixed_Point_Definition,       -- 3.5.9(6)

   An_Unconstrained_Array_Definition,      -- 3.6(2)
   A_Constrained_Array_Definition,         -- 3.6(2)

   A_Record_Type_Definition,               -- 3.8(2)
   A_Tagged_Record_Type_Definition,        -- 3.8(2)
   An_Interface_Type_Definition,           -- 3.9.4 (2)

   An_Access_Type_Definition);             -- 3.10(2)   -> Access_Type_Kinds
```

The comments list a reference to the definition in the Ada Standard for each kind of type; the subordinate kind (if any) is given as well.

3.7.11 type Formal_Type_Kinds

```
type Formal_Type_Kinds is (

   Not_A_Formal_Type_Definition,           -- An unexpected element

   A_Formal_Private_Type_Definition,       -- 12.5.1(2)
   A_Formal_Tagged_Private_Type_Definition, -- 12.5.1(2)

   A_Formal_Derived_Type_Definition,       -- 12.5.1(3)

   A_Formal_Discrete_Type_Definition,      -- 12.5.2(2)

   A_Formal_Signed_Integer_Type_Definition, -- 12.5.2(3)
   A_Formal_Modular_Type_Definition,       -- 12.5.2(4)

   A_Formal_Floating_Point_Definition,     -- 12.5.2(5)

   A_Formal_Ordinary_Fixed_Point_Definition, -- 12.5.2(6)
   A_Formal_Decimal_Fixed_Point_Definition, -- 12.5.2(7)

   A_Formal_Unconstrained_Array_Definition, -- 12.5.3(2), 3.6(3)
   A_Formal_Constrained_Array_Definition,  -- 12.5.3(2), 3.6(5)

   A_Formal_Access_Type_Definition,        -- 12.5.4(2)
                                           --      -> Access_Type_Kinds
   A_Formal_Interface_Type_Definition);    -- 12.5.5 (2)
```

The comments list a reference to the definition in the Ada Standard for each kind of formal type; the subordinate kind (if any) is given as well.

3.7.12 type Access_Type_Kinds

```
type Access_Type_Kinds is (              -- 3.10 in 8652:1995
   Not_An_Access_Type_Definition,        -- An unexpected element
   A_Pool_Specific_Access_To_Variable,   -- access subtype_indication
   An_Access_To_Variable,                -- access all subtype_indication
   An_Access_To_Constant,                -- access constant subtype_indication
   An_Access_To_Procedure,               -- access procedure
   An_Access_To_Protected_Procedure,     -- access protected procedure
   An_Access_To_Function,                -- access function
   An_Access_To_Protected_Function);     -- access protected function
```

The comments list a reference to the definition in the Ada Standard for each kind of access type.

The following Access_Type_Kinds subtypes are not used by ASIS but are provided for the convenience of the ASIS user:

```
subtype Access_To_Object_Definition is Access_Type_Kinds range
          A_Pool_Specific_Access_To_Variable .. An_Access_To_Constant;

subtype Access_To_Subprogram_Definition is Access_Type_Kinds range
          An_Access_To_Procedure .. An_Access_To_Protected_Function;
```

3.7.13 type Access_Definition_Kinds

```
type Access_Definition_Kinds is (    -- 3.3.1(2) / 3.10(6)
   Not_An_Access_Definition,                    -- An unexpected element
   An_Anonymous_Access_To_Variable,             -- 3.3.1(2) [...] access subtype_mark
   An_Anonymous_Access_To_Constant,             -- 3.3.1(2) / 3.10(6)
                                                --   [...] access constant subtype_mark
   An_Anonymous_Access_To_Procedure,            -- 3.10(6) access procedure
   An_Anonymous_Access_To_Protected_Procedure,  -- 3.10(6) access protected procedure
   An_Anonymous_Access_To_Function,             -- 3.10(6) access function
   An_Anonymous_Access_To_Protected_Function);  -- 3.10(6) access protected function

subtype An_Anonymous_Access_to_Object_Definition is Access_Definition_Kinds
   range An_Anonymous_Access_To_Variable .. An_Anonymous_Access_To_Constant;

subtype An_Anonymous_Access_to_Subprogram_Definition is Access_Definition_Kinds
   range An_Anonymous_Access_To_Procedure ..
   An_Anonymous_Access_To_Protected_Function;
```

3.7.14 type Root_Type_Kinds

```
type Root_Type_Kinds is (
   Not_A_Root_Type_Definition,          -- An unexpected element
   A_Root_Integer_Definition,           -- 3.4.1(8)
   A_Root_Real_Definition,              -- 3.4.1(8)
   A_Universal_Integer_Definition,      -- 3.4.1(6)
   A_Universal_Real_Definition,         -- 3.4.1(6)
   A_Universal_Fixed_Definition,        -- 3.4.1(6)
   A_Universal_Access_Definition);      -- 3.4.1(6)
```

The comments list a reference to the definition in the Ada Standard for each kind of root or universal type.

3.7.15 type Constraint_Kinds

```
type Constraint_Kinds is (
   Not_A_Constraint,                    -- An unexpected element
```

```
   A_Range_Attribute_Reference,          -- 3.5(2)
   A_Simple_Expression_Range,            -- 3.2.2, 3.5(3)
   A_Digits_Constraint,                  -- 3.2.2, 3.5.9
   A_Delta_Constraint,                   -- 3.2.2, J.3
   An_Index_Constraint,                  -- 3.2.2, 3.6.1
   A_Discriminant_Constraint);           -- 3.2.2
```

The comments list a reference to the definition in the Ada Standard for each kind of constraint or range.

3.7.16 type Discrete_Range_Kinds

```
type Discrete_Range_Kinds is (
   Not_A_Discrete_Range,                 -- An unexpected element

   A_Discrete_Subtype_Indication,        -- 3.6.1(6), 3.2.2
   A_Discrete_Range_Attribute_Reference, -- 3.6.1, 3.5
   A_Discrete_Simple_Expression_Range);  -- 3.6.1, 3.5
```

The comments list a reference to the definition in the Ada Standard for each kind of range.

3.7.17 type Interface_Kinds

```
type Interface_Kinds is (  -- 3.9.4 (2)
   Not_An_Interface,                  -- An unexpected element

   An_Ordinary_Interface,             -- 3.9.4(2)
   A_Limited_Interface,               -- 3.9.4(2)
   A_Task_Interface,                  -- 3.9.4(2)
   A_Protected_Interface,             -- 3.9.4(2)
   A_Synchronized_Interface);         -- 3.9.4(2)
```

The comments list a reference to the definition in the Ada Standard for each kind of interface.

3.7.18 type Association_Kinds

```
type Association_Kinds is (
   Not_An_Association,                -- An unexpected element

   A_Pragma_Argument_Association,     -- 2.8
   A_Discriminant_Association,        -- 3.7.1
   A_Record_Component_Association,    -- 4.3.1
   An_Array_Component_Association,    -- 4.3.3
   A_Parameter_Association,           -- 6.4
   A_Generic_Association);            -- 12.3
```

The comments list a reference to the definition in the Ada Standard for each kind of association.

3.7.19 type Expression_Kinds

Type Expression_Kinds describes general expression classifications.

```
type Expression_Kinds is (
   Not_An_Expression,                 -- An unexpected element

   A_Box_Expression,                  -- 4.3.1(4), 4.3.3(3,6)

   An_Integer_Literal,                -- 2.4
   A_Real_Literal,                    -- 2.4.1
   A_String_Literal,                  -- 2.6
```

```
   An_Identifier,                          -- 4.1
   An_Operator_Symbol,                     -- 4.1
   A_Character_Literal,                    -- 4.1
   An_Enumeration_Literal,                 -- 4.1
   An_Explicit_Dereference,                -- 4.1
   An_Implicit_Dereference,                -- 4.1
   A_Function_Call,                        -- 4.1

   An_Indexed_Component,                   -- 4.1.1
   A_Slice,                                -- 4.1.2
   A_Selected_Component,                   -- 4.1.3
   An_Attribute_Reference,                 -- 4.1.4 -> Attribute_Kinds

   A_Record_Aggregate,                     -- 4.3
   An_Extension_Aggregate,                 -- 4.3
   A_Positional_Array_Aggregate,           -- 4.3
   A_Named_Array_Aggregate,                -- 4.3

   An_And_Then_Short_Circuit,              -- 4.4
   An_Or_Else_Short_Circuit,               -- 4.4

   An_In_Range_Membership_Test,            -- 4.4
   A_Not_In_Range_Membership_Test,         -- 4.4
   An_In_Type_Membership_Test,             -- 4.4
   A_Not_In_Type_Membership_Test,          -- 4.4

   A_Null_Literal,                         -- 4.4
   A_Parenthesized_Expression,             -- 4.4

   A_Type_Conversion,                      -- 4.6
   A_Qualified_Expression,                 -- 4.7

   An_Allocation_From_Subtype,             -- 4.8
   An_Allocation_From_Qualified_Expression); -- 4.8
```

The comments list a reference to the definition in the Ada Standard for each kind of expression; the subordinate kind (if any) is given as well.

3.7.20 type Operator_Kinds

Type Operator_Kinds describes classification of the various Ada predefined operators.

```
type Operator_Kinds is (               -- 4.5 in the Ada Standard
   Not_An_Operator,                    -- An unexpected element

   An_And_Operator,                    -- and
   An_Or_Operator,                     -- or
   An_Xor_Operator,                    -- xor
   An_Equal_Operator,                  -- =
   A_Not_Equal_Operator,               -- /=
   A_Less_Than_Operator,               -- <
   A_Less_Than_Or_Equal_Operator,      -- <=
   A_Greater_Than_Operator,            -- >
   A_Greater_Than_Or_Equal_Operator,   -- >=
   A_Plus_Operator,                    -- +
   A_Minus_Operator,                   -- -
   A_Concatenate_Operator,             -- &
   A_Unary_Plus_Operator,              -- +
   A_Unary_Minus_Operator,             -- -
   A_Multiply_Operator,                -- *
   A_Divide_Operator,                  -- /
   A_Mod_Operator,                     -- mod
   A_Rem_Operator,                     -- rem
   An_Exponentiate_Operator,           -- **
   An_Abs_Operator,                    -- abs
   A_Not_Operator);                    -- not
```

3.7.21 type Attribute_Kinds

Type Attribute_Kinds describes classifications of Ada attributes.

```
type Attribute_Kinds is (
   Not_An_Attribute,                -- An unexpected element

   An_Access_Attribute,             -- 3.10.2(24), 3.10.2(32), K(2), K(4)
   An_Address_Attribute,            -- 13.3(11), J.7.1(5), K(6)
   An_Adjacent_Attribute,           -- A.5.3(48), K(8)
   An_Aft_Attribute,                -- 3.5.10(5), K(12)
   An_Alignment_Attribute,          -- 13.3(23), K(14)

   A_Base_Attribute,                -- 3.5(15), K(17)
   A_Bit_Order_Attribute,           -- 13.5.3(4), K(19)
   A_Body_Version_Attribute,        -- E.3(4), K(21)

   A_Callable_Attribute,            -- 9.9(2), K(23)
   A_Caller_Attribute,              -- C.7.1(14), K(25)
   A_Ceiling_Attribute,             -- A.5.3(33), K(27)
   A_Class_Attribute,               -- 3.9(14), 7.3.1(9), K(31), K(34)
   A_Component_Size_Attribute,      -- 13.3(69), K(36)
   A_Compose_Attribute,             -- A.5.3(24), K(38)
   A_Constrained_Attribute,         -- 3.7.2(3), J.4(2), K(42)
   A_Copy_Sign_Attribute,           -- A.5.3(51), K(44)
   A_Count_Attribute,               -- 9.9(5), K(48)

   A_Definite_Attribute,            -- 12.5.1(23), K(50)
   A_Delta_Attribute,               -- 3.5.10(3), K(52)
   A_Denorm_Attribute,              -- A.5.3(9), K(54)
   A_Digits_Attribute,              -- 3.5.8(2), 3.5.10(7), K(56), K(58)

   An_Exponent_Attribute,           -- A.5.3(18), K(60)
   An_External_Tag_Attribute,       -- 13.3(75), K(64)

   A_First_Attribute,               -- 3.5(12), 3.6.2(3), K(68), K(70)
   A_First_Bit_Attribute,           -- 13.5.2(3), K(72)
   A_Floor_Attribute,               -- A.5.3(30), K(74)
   A_Fore_Attribute,                -- 3.5.10(4), K(78)
   A_Fraction_Attribute,            -- A.5.3(21), K(80)

   An_Identity_Attribute,           -- 11.4.1(9), C.7.1(12), K(84), K(86)
   An_Image_Attribute,              -- 3.5(35), K(88)
   An_Input_Attribute,              -- 13.13.2(22), 13.13.2(32), K(92), K(96)

   A_Last_Attribute,                -- 3.5(13), 3.6.2(5), K(102), K(104)
   A_Last_Bit_Attribute,            -- 13.5.2(4), K(106)
   A_Leading_Part_Attribute,        -- A.5.3(54), K(108)
   A_Length_Attribute,              -- 3.6.2(9), K(117)

   A_Machine_Attribute,             -- A.5.3(60), K(119)
   A_Machine_Emax_Attribute,        -- A.5.3(8), K(123)
   A_Machine_Emin_Attribute,        -- A.5.3(7), K(125)
   A_Machine_Mantissa_Attribute,    -- A.5.3(6), K(127)
   A_Machine_Overflows_Attribute,   -- A.5.3(12), A.5.4(4), K(129), K(131)
   A_Machine_Radix_Attribute,       -- A.5.3(2), A.5.4(2), K(133), K(135)
   A_Machine_Rounding_Attribute,    -- A.5.3(41.1/2)), K(135.1/2)
   A_Machine_Rounds_Attribute,      -- A.5.3(11), A.5.4(3), K(137), K(139)
   A_Max_Attribute,                 -- 3.5(19), K(141)
   A_Max_Size_In_Storage_Elements_Attribute,-- 13.11.1(3), K(145)
   A_Min_Attribute,                 -- 3.5(16), K(147)
   A_Mod_Attribute,                 -- 3.5.4(16.3/2)), K(150.1/2)
   A_Model_Attribute,               -- A.5.3(68), G.2.2(7), K(151)
   A_Model_Emin_Attribute,          -- A.5.3(65), G.2.2(4), K(155)
   A_Model_Epsilon_Attribute,       -- A.5.3(66), K(157)
   A_Model_Mantissa_Attribute,      -- A.5.3(64), G.2.2(3), K(159)
   A_Model_Small_Attribute,         -- A.5.3(67), K(161)
   A_Modulus_Attribute,             -- 3.5.4(17), K(163)

   An_Output_Attribute,             -- 13.13.2(19), 13.13.2(29), K(165), K(169)

   A_Partition_ID_Attribute,        -- E.1(9), K(173)
   A_Pos_Attribute,                 -- 3.5.5(2), K(175)
   A_Position_Attribute,            -- 13.5.2(2), K(179)
   A_Pred_Attribute,                -- 3.5(25), K(181)
   A_Priority_Attribute,            -- D.2.6(27/2)), K(184.1/2)
```

```
   A_Range_Attribute,              -- 3.5(14), 3.6.2(7), K(187), K(189)
   A_Read_Attribute,               -- 13.13.2(6), 13.13.2(14), K(191), K(195)
   A_Remainder_Attribute,          -- A.5.3(45), K(199)
   A_Round_Attribute,              -- 3.5.10(12), K(203)
   A_Rounding_Attribute,           -- A.5.3(36), K(207)

   A_Safe_First_Attribute,         -- A.5.3(71), G.2.2(5), K(211)
   A_Safe_Last_Attribute,          -- A.5.3(72), G.2.2(6), K(213)
   A_Scale_Attribute,              -- 3.5.10(11), K(215)
   A_Scaling_Attribute,            -- A.5.3(27), K(217)
   A_Signed_Zeros_Attribute,       -- A.5.3(13), K(221)
   A_Size_Attribute,               -- 13.3(40), 13.3(45), K(223), K(228)
   A_Small_Attribute,              -- 3.5.10(2), K(230)
   A_Stream_Size_Attribute,        -- 13.13.2(1.2/2)), K(237.1/2)
   A_Storage_Pool_Attribute,       -- 13.11(13), K(232)
   A_Storage_Size_Attribute,       -- 13.3(60), 13.11(14), J.9(2), K(234),
                                   -- K(236)
   A_Succ_Attribute,               -- 3.5(22), K(238)

   A_Tag_Attribute,                -- 3.9(16), 3.9(18), K(242), K(244)
   A_Terminated_Attribute,         -- 9.9(3), K(246)
   A_Truncation_Attribute,         -- A.5.3(42), K(248)

   An_Unbiased_Rounding_Attribute,-- A.5.3(39), K(252)
   An_Unchecked_Access_Attribute, -- 13.10(3), H.4(18), K(256)

   A_Val_Attribute,                -- 3.5.5(5), K(258)
   A_Valid_Attribute,              -- 13.9.2(3), H(6), K(262)
   A_Value_Attribute,              -- 3.5(52), K(264)
   A_Version_Attribute,            -- E.3(3), K(268)

   A_Wide_Image_Attribute,         -- 3.5(28), K(270)
   A_Wide_Value_Attribute,         -- 3.5(40), K(274)
   A_Wide_Wide_Image_Attribute,    -- 3.5(27.1/2)), K(277.1/2)
   A_Wide_Wide_Value_Attribute,    -- 3.5(39.1/2)), K(277.5/2)
   A_Wide_Wide_Width_Attribute,    -- 3.5(37.1/2)), K(277.9/2)
   A_Wide_Width_Attribute,         -- 3.5(38), K(278)
   A_Width_Attribute,              -- 3.5(39), K(280)
   A_Write_Attribute,              -- 13.13.2(3), 13.13.2(11), K(282), K(286)

   An_Implementation_Defined_Attribute,  -- Ada Standard, Annex M

   An_Unknown_Attribute);          -- Unknown to ASIS
```

The comments list a reference to the definition in the Ada Standard for each attribute.

3.7.22 type Statement_Kinds

Type Statement_Kinds describes classifications of Ada statements.

```
type Statement_Kinds is (
   Not_A_Statement,                -- An unexpected element

   A_Null_Statement,               -- 5.1
   An_Assignment_Statement,        -- 5.2
   An_If_Statement,                -- 5.3
   A_Case_Statement,               -- 5.4

   A_Loop_Statement,               -- 5.5
   A_While_Loop_Statement,         -- 5.5
   A_For_Loop_Statement,           -- 5.5
   A_Block_Statement,              -- 5.6
   An_Exit_Statement,              -- 5.7
   A_Goto_Statement,               -- 5.8

   A_Procedure_Call_Statement,     -- 6.4
   A_Simple_Return_Statement,      -- 6.5
   An_Extended_Return_Statement,   -- 6.5

   An_Accept_Statement,            -- 9.5.2
   An_Entry_Call_Statement,        -- 9.5.3

   A_Requeue_Statement,            -- 9.5.4
   A_Requeue_Statement_With_Abort, -- 9.5.4

   A_Delay_Until_Statement,        -- 9.6
   A_Delay_Relative_Statement,     -- 9.6
```

```
   A_Terminate_Alternative_Statement,    -- 9.7.1
   A_Selective_Accept_Statement,         -- 9.7.1
   A_Timed_Entry_Call_Statement,         -- 9.7.2
   A_Conditional_Entry_Call_Statement,   -- 9.7.3
   An_Asynchronous_Select_Statement,     -- 9.7.4

   An_Abort_Statement,                   -- 9.8
   A_Raise_Statement,                    -- 11.3
   A_Code_Statement);                    -- 13.8

A_Return_Statement : Statement_Kinds renames A_Simple_Return_Statement;
-- For compatibility with a prior version of this Standard
```

The comments list a reference to the definition in the Ada Standard for each statement.

3.7.23 type Path_Kinds

A_Path elements represent execution path alternatives presented by the if_statement, case_statement, and the four forms of select_statement. Each statement path alternative encloses component elements that represent a sequence_of_statements. Some forms of A_Path elements also have as a component elements that represent a condition, an optional guard, or a discrete_choice_list.

ASIS treats the select_alternative, entry_call_alternative, and triggering_alternative, as the syntactic equivalent of a sequence_of_statements. Specifically, the terminate_alternative (terminate;) is treated as the syntactical equivalent of a single statement and are represented as Statement_Kinds'A_Terminate_Alternative_Statement. This allows queries to directly provide the sequence_of_statements enclosed by A_Path elements, avoiding the extra step of returning an element representing such an alternative.

For example,

```
select    -- A_Select_Path enclosing a sequence of two statements
   accept Next_Work_Item(WI : in Work_Item) do
      Current_Work_Item := WI;
   end;
   Process_Work_Item(Current_Work_Item);
or        -- An_Or_Path enclosing a guard and a sequence of two statements
   when Done_Early =>
      accept Shut_Down;
      exit;
or        -- An_Or_Path enclosing a sequence with only a single statement
   terminate;
end select;
```

The type definition is:

```
type Path_Kinds is (
   Not_A_Path,                    -- An unexpected element
   An_If_Path,                    -- 5.3:
                                  --    if condition then
                                  --       sequence_of_statements
   An_Elsif_Path,                 -- 5.3:
                                  --    elsif condition then
                                  --       sequence_of_statements
   An_Else_Path,                  -- 5.3, 9.7.1, 9.7.3:
                                  --    else sequence_of_statements
   A_Case_Path,                   -- 5.4:
                                  --    when discrete_choice_list =>
                                  --       sequence_of_statements
```

```
   A_Select_Path,                -- 9.7.1:
                                 --    select [guard] select_alternative
                                 -- 9.7.2, 9.7.3:
                                 --    select entry_call_alternative
                                 -- 9.7.4:
                                 --    select triggering_alternative
   An_Or_Path,                   -- 9.7.1:
                                 --    or [guard] select_alternative
                                 -- 9.7.2:
                                 --    or delay_alternative
   A_Then_Abort_Path);           -- 9.7.4
                                 --    then abort sequence_of_statements
```

The comments list a reference to the definition in the Ada Standard for each path, and the path represented.

3.7.24 type Clause_Kinds

Type Clause_Kinds describes kinds of clauses.

```
type Clause_Kinds is (
   Not_A_Clause,              -- An unexpected element
   A_Use_Package_Clause,      -- 8.4
   A_Use_Type_Clause,         -- 8.4
   A_With_Clause,             -- 10.1.2
   An_Aspect_Clause           -- 13.1    -> Aspect_Clause_Kinds
   A_Component_Clause);       -- 13.5.1
```

The comments list a reference to the definition in the Ada Standard for each clause, and any subordinate kinds. Note that a private with clause is indicated by the Has_Private query (see 13.13). Similarly, a limited with clause is indicated by the Has_Limited query (see 13.12).

3.7.25 type Aspect_Clause_Kinds

Type Aspect_Clause_Kinds describes varieties of aspect clauses.

```
type Aspect_Clause_Kinds is (
   Not_An_Aspect_Clause,             -- An unexpected element
   An_Attribute_Definition_Clause,   -- 13.3
   An_Enumeration_Representation_Clause,  -- 13.4
   A_Record_Representation_Clause,   -- 13.5.1
   An_At_Clause);                    -- J.7
```

The comments list a reference to the definition in the Ada Standard for each aspect clause.

3.8 type Compilation_Unit

The type Compilation_Unit is used to represent an Ada *Compilation Unit*.

The text of a program is submitted to the compiler in one or more compilations. Each compilation is a succession of compilation units.

Compilation units are composed of three distinct parts:

a) A context clause.

b) The declaration of a library_item or unit.

c) Pragmas that apply to the compilation, of which the unit is a part.

The context clause contains zero or more with clauses, use clauses, and pragmas.

ASIS treats Pragmas that appear immediately after the context clause and before the subsequent declaration part as belonging to the context clause part.

The declaration associated with a compilation unit is one of: a package, a procedure, a function, a generic, or a subunit for normal units. The associated declaration is a Nil_Element for An_Unknown_Unit and Nonexistent units.

The type Compilation_Unit is a handle for compilation units as a whole. An object of the type Compilation_Unit deals with the external view of compilation units such as their relationships with other units or their compilation attributes.

Compilation_Unit shall be an undiscriminated private type.

```
type Compilation_Unit is private;
Nil_Compilation_Unit : constant Compilation_Unit;

function "=" (Left  : in Compilation_Unit;
              Right : in Compilation_Unit)
              return Boolean is abstract;
```

3.9 type Compilation_Unit_List

```
type Compilation_Unit_List is
       array (List_Index range <>) of Compilation_Unit;

Nil_Compilation_Unit_List : constant Compilation_Unit_List;
```

3.10 Unit Kinds

Unit Kinds are enumeration types describing the various kinds of units. These element kinds are used within package Asis.Compilation_Units.

3.10.1 type Unit_Kinds

Unit_Kinds defines the varieties of compilation units which form compilations, including compilations having no compilation units but consisting of configuration pragmas or comments.

```
type Unit_Kinds is (

   Not_A_Unit,                    -- A Nil_Compilation_Unit

   A_Procedure,
   A_Function,
   A_Package,

   A_Generic_Procedure,
   A_Generic_Function,
   A_Generic_Package,

   A_Procedure_Instance,
   A_Function_Instance,
   A_Package_Instance,

   A_Procedure_Renaming,
   A_Function_Renaming,
   A_Package_Renaming,

   A_Generic_Procedure_Renaming,
   A_Generic_Function_Renaming,
   A_Generic_Package_Renaming,

   A_Procedure_Body,      -- A unit interpreted only as the completion
                          -- of a procedure, or a unit interpreted as
                          -- both the declaration and body of a library
                          -- procedure. Ada Standard 10.1.4(4)

   A_Function_Body,       -- A unit interpreted only as the completion
                          -- of a function, or a unit interpreted as
                          -- both the declaration and body of a library
                          -- function. Ada Standard 10.1.4(4)

   A_Package_Body,
```

```
   A_Procedure_Body_Subunit,
   A_Function_Body_Subunit,
   A_Package_Body_Subunit,
   A_Task_Body_Subunit,
   A_Protected_Body_Subunit,
   A_Nonexistent_Declaration,    -- A unit that does not exist but is:
                                 -- 1) mentioned in a with clause of
                                 --    another unit or,
                                 -- 2) a required corresponding
                                 --    library_unit_declaration
   A_Nonexistent_Body,           -- A unit that does not exist but is:
                                 -- 1) known to be a corresponding
                                 --    subunit or,
                                 -- 2) a required corresponding
                                 --    library_unit_body
   A_Configuration_Compilation, -- Corresponds to the whole content of a
                                 -- compilation with no compilation_unit,
                                 -- but possibly containing comments,
                                 -- configuration pragmas, or both.
                                 -- A Context may have any number of
                                 -- units of A_Configuration_Compilation kind.
                                 -- A unit of A_Configuration_Compilation
                                 -- does not have a name. This unit
                                 -- represents configuration pragmas that
                                 -- are "in effect". The only interface that
                                 -- returns this unit kind is
                                 -- Enclosing_Compilation_Unit when given
                                 -- A_Pragma element obtained from Configuration_Pragmas.
   An_Unknown_Unit);             -- An indeterminable or proprietary unit

subtype A_Subprogram_Declaration is Unit_Kinds range
            A_Procedure ..
            A_Function;

subtype A_Subprogram_Renaming is Unit_Kinds range
            A_Procedure_Renaming ..
            A_Function_Renaming;

subtype A_Generic_Unit_Declaration is Unit_Kinds range
            A_Generic_Procedure ..
            A_Generic_Package;

subtype A_Generic_Unit_Instance is Unit_Kinds range
            A_Procedure_Instance ..
            A_Package_Instance;

subtype A_Subprogram_Body is Unit_Kinds range
            A_Procedure_Body ..
            A_Function_Body;

subtype A_Library_Unit_Body is Unit_Kinds range
            A_Procedure_Body ..
            A_Package_Body;

subtype A_Generic_Renaming is Unit_Kinds range
            A_Generic_Procedure_Renaming ..
            A_Generic_Package_Renaming;

subtype A_Renaming is Unit_Kinds range
            A_Procedure_Renaming ..
            A_Generic_Package_Renaming;

subtype A_Subunit is Unit_Kinds range
            A_Procedure_Body_Subunit ..
            A_Protected_Body_Subunit;
```

3.10.2 type Unit_Classes

Unit_Classes defines a classification of compilation units according to whether they are public or private, declaration or body, and library unit or subunit.

```
type Unit_Classes is (   -- Ada Standard 10.1.1(12), 10.1.3
   Not_A_Class,              -- A nil, nonexistent, unknown,
                             -- or configuration compilation unit class.
   A_Public_Declaration,     -- library_unit_declaration or
                             -- library_unit_renaming_declaration.
   A_Public_Body,            -- library_unit_body interpreted only as a
                             -- completion. Its declaration is public.
   A_Public_Declaration_And_Body,
                             -- subprogram_body interpreted as both a
                             -- declaration and body of a library
                             -- subprogram - Ada Standard 10.1.4(4).
   A_Private_Declaration,    -- private library_unit_declaration or
                             -- private library_unit_renaming_declaration.
   A_Private_Body,           -- library_unit_body interpreted only as a
                             -- completion. Its declaration is private.
   A_Separate_Body);         -- separate (parent_unit_name) proper_body.
```

3.10.3 type Unit_Origins

Unit_Origins defines classification of possible unit origination.

```
type Unit_Origins is (
   Not_An_Origin,         -- A nil or nonexistent unit origin
                          -- An_Unknown_Unit can be any origin
   A_Predefined_Unit,     -- Ada predefined language environment units
                          -- listed in Annex A(2). These include
                          -- Standard and the three root library
                          -- units: Ada, Interfaces, and System,
                          -- and their descendants. i.e., Ada.Text_Io,
                          -- Ada.Calendar, Interfaces.C, etc.
   An_Implementation_Unit,
                          -- Implementation specific library units,
                          -- e.g., runtime support packages, utility
                          -- libraries, etc. It is not required
                          -- that any implementation supplied units
                          -- have this origin. This is a suggestion.
                          -- Implementations might provide, for
                          -- example, precompiled versions of public
                          -- domain software that could have
                          -- An_Application_Unit origin.
   An_Application_Unit);  -- Neither A_Predefined_Unit or
                          -- An_Implementation_Unit
```

3.10.4 type Relation_Kinds

Relation_Kinds defines classification of unit relationships.

```
type Relation_Kinds is (
   Ancestors,
   Descendants,
   Supporters,
   Dependents,
   Family,
   Needed_Units);
```

- *Ancestors* of a library unit are itself, its parent, its parent's parent, and so on. (Standard is an ancestor of every library unit).

- The *Descendants* relation is the inverse of the ancestor relation. Ada Standard 10.1.1(11).
- *Supporters* of a compilation unit are units on which it semantically depends. Ada Standard 10.1.1(26).

  The Supporters relation is transitive; units that are supporters of library units mentioned in a with clause of a compilation unit are also supporters of that compilation unit.

  A parent declaration is a Supporter of its descendant units.

  Each library unit mentioned in the with clauses of a compilation unit is a Supporter of that compilation unit and (recursively) any completion, child units, or subunits that are included in the declarative region of that compilation unit. Ada Standard 8.1(7-10).

  A library_unit_body has as a Supporter, its corresponding library_unit_declaration, if any.

  The parent body of a subunit is a Supporter of the subunit.
- *Dependents* of a compilation unit are all the compilation units that depend semantically on it.

  The Dependents relation is transitive; Dependents of a unit include the unit's Dependents, each dependent unit's Dependents, and so on. A unit that is a dependent of a compilation unit also is a dependent of the compilation unit's Supporters.

  Child units are Dependents of their ancestor units.

  A compilation unit that mentions other library units in its with clauses is one of the Dependents of those library units.

  A library_unit_body is a Dependent of its corresponding library_unit_declaration, if any.

  A subunit is a Dependent of its parent body.

  A compilation unit that contains an attribute_reference of a type defined in another compilation unit is a Dependent of the other unit.

  For example:

  If A mentions B and B mentions C in with_clauses
  then A directly depends on B,
  B directly depends on C,
  A indirectly depends on C, and
  both A and B are dependents of C.

  Dependencies between compilation units may also be introduced by inline expansions (Ada Standard 10.1.4(7)) and for certain other compiler optimizations. These relations are intended to reflect all of these considerations.
- The *family* of a given unit is defined as the set of compilation units that comprise the given unit's declaration, body, descendants, and subunits (and subunits of subunits and descendants, etc.).
- The *needed units* of a given unit is defined as the set of all the Ada units ultimately needed by that unit to form a partition. Ada Standard 10.2(2-7).

  The term *closure* is commonly used with similar meaning.

  For example:

  Assume the body of C has a subunit C.S and the declaration of C has child units C.Y and C.Z.

If A mentions B and B mentions C and C.Y in with_clauses and C does not have a with_clause, then the needed units of A are:
library unit declaration C
child library unit declaration C.Y
child library unit body C.Y, if any
library unit body C
subunit C.S
library unit declaration B
library unit body B, if any
library unit declaration A
library unit body A, if any

Child unit C.Z is only part of the Needed_Units if it is needed according to the rule s of the Ada standard, 10.2(2-7).

3.11 type Traverse_Control

Traverse_Control defines controls for the traversal generic provided in package Asis.Iterator. It is defined in package Asis to facilitate automatic translation to IDL (See Annex C for details).

```
type Traverse_Control is (
   Continue,               -- Continues the normal depth-first traversal.
   Abandon_Children,       -- Prevents traversal of the current element's
                           -- children.
   Abandon_Siblings,       -- Prevents traversal of the current element's
                           -- children and remaining siblings.
   Terminate_Immediately); -- Does exactly that.
```

3.12 type Program_Text

Program_Text provides an abstraction for the program text for a program's source code. This is the return type for all Image functions.

```
subtype Program_Text is Wide_String;
```

Section 4: package Asis.Errors

The library package Asis.Errors shall exist. The package shall provide interfaces equivalent to those described in the following subclauses.

ASIS reports all operational errors by raising an exception. Whenever an ASIS implementation raises one of the exceptions declared in package Asis.Exceptions, it will previously have set the values returned by the Status and Diagnosis queries to indicate the cause of the error. The possible values for Status are indicated in the definition of Error_Kinds below, with suggestions for the associated contents of the Diagnosis string as a comment.

The Diagnosis and Status queries are provided in the Asis.Implementation package to supply more information about the reasons for raising any exception.

ASIS applications are encouraged to follow this same convention whenever they explicitly raise any ASIS exception--always record a Status and Diagnosis prior to raising the exception.

4.1 type Error_Kinds

This enumeration type describes the various kinds of errors.

```
type Error_Kinds is (
    Not_An_Error,                   -- No error is presently recorded

    Value_Error,                    -- Routine argument value invalid
    Initialization_Error,           -- ASIS is uninitialized
    Environment_Error,              -- ASIS could not initialize
    Parameter_Error,                -- Bad Parameter given to Initialize
    Capacity_Error,                 -- Implementation overloaded
    Name_Error,                     -- Context/unit not found
    Use_Error,                      -- Context/unit not use/open-able
    Data_Error,                     -- Context/unit bad/invalid/corrupt
    Text_Error,                     -- The program text cannot be located
    Storage_Error,                  -- Storage_Error suppressed
    Obsolete_Reference_Error,       -- Argument or result is invalid due to
                                    -- and inconsistent compilation unit
    Unhandled_Exception_Error,      -- Unexpected exception suppressed
    Not_Implemented_Error,          -- Functionality not implemented
    Internal_Error);                -- Implementation internal failure
```

Section 5: package Asis.Exceptions

The library package Asis.Exceptions shall exist. The package shall provide interfaces equivalent to those described in this clause.

ASIS exceptions are:

```
ASIS_Inappropriate_Context : exception;
```

Raised when ASIS is passed a Context value that is not appropriate for the operation. This exception will typically indicate that a user error has occurred within the application.

```
ASIS_Inappropriate_Container : exception;
```

Raised when ASIS is passed a Container value that is not appropriate for the operation. This exception will typically indicate that a user error has occurred within the application.

```
ASIS_Inappropriate_Compilation_Unit : exception;
```

Raised when ASIS is passed a Compilation_Unit value that is not appropriate. This exception will typically indicate that a user error has occurred within the application.

```
ASIS_Inappropriate_Element : exception;
```

Raised when ASIS is given an Element value that is not appropriate. This exception will typically indicate that a user error has occurred within the application.

```
ASIS_Inappropriate_Line : exception;
```

Raised when ASIS is given a Line value that is not appropriate.

```
ASIS_Inappropriate_Line_Number : exception;
```

Raised when ASIS is given a Line_Number value that is not appropriate. This exception will typically indicate that a user error has occurred within the application.

```
ASIS_Inappropriate_View : exception;
```

Raised when ASIS is given an inappropriate view or profile.

```
ASIS_Not_In_Context : exception;
```

Raised for any operation defined in Asis.Views, Asis.Program_Units, Asis.Subtype_Views, Asis.Object_Views, Asis.Profiles, Asis.Callable_Views, Asis.Package_Views, Asis.Generic_Views, Asis.Exception_Views, Asis.Statement_Views, and their subpackages and children, if the result is an entity that is not part of the current context.

```
ASIS_Failed : exception;
```

This is a catch-all exception that may be raised for different reasons in different ASIS implementations. All ASIS routines may raise ASIS_Failed whenever they cannot normally complete their operation. This exception will typically indicate a failure of the underlying ASIS implementation.

NOTE ASIS_Not_In_Context can happen, for instance, if the View is of an incomplete view, and the full view is not part of the context. It also can happen if the context includes only a portion of the semantic dependencies of the queried unit.

Section 6: package Asis.Implementation

The library package Asis.Implementation shall exist. The package shall provide interfaces equivalent to those described in the following subclauses.

Asis.Implementation provides queries to initialize, finalize, and query the error status of the ASIS Implementation.

6.1 function ASIS_Version

```
function ASIS_Version return Wide_String;
```

Returns a value which identifies the version of the ASIS interface, e.g., "2.1"

6.2 function ASIS_Implementor

```
function ASIS_Implementor return Wide_String;
```

Returns a value which identifies the name of the implementor, e.g., "Ada Inc."

6.3 function ASIS_Implementor_Version

```
function ASIS_Implementor_Version return Wide_String;
```

Returns a value which identifies the implementation's version, e.g., "5.2a"

6.4 function ASIS_Implementor_Information

```
function ASIS_Implementor_Information return Wide_String;
```

Returns a value which identifies additional implementation information, e.g., "Copyright ..."

6.5 function Is_Initialized

```
function Is_Initialized return Boolean;
```

Returns True if ASIS is currently initialized, and returns False otherwise.

6.6 procedure Initialize

```
procedure Initialize (Parameters : in Wide_String := "");
```

Parameters specify implementation specific parameters.

Performs any necessary initialization activities. This shall be invoked at least once before any other ASIS services are used. Parameter values are implementation dependent. The call is ignored if ASIS is already initialized. All ASIS queries and services are ready for use once this call completes.

Raises ASIS_Failed if ASIS failed to initialize or if the Parameters argument is invalid. Status is Environment_Error or Parameter_Error.

NOTE The ASIS implementation may be Initialized and Finalized any number of times during the operation of an ASIS program. However, all existing Context, Compilation_Unit and Element values become invalid when ASIS Is_Finalized. Subsequent calls to ASIS queries or services using such invalid Compilation_Unit or Element values will cause ASIS_Inappropriate_Context to be raised.

6.7 function Is_Finalized

```
function Is_Finalized return Boolean;
```

Returns True if ASIS is currently finalized or if ASIS has never been initialized, and returns False otherwise.

6.8 procedure Finalize

```
procedure Finalize (Parameters : in Wide_String := "");
```

Parameters specify any implementation required parameter values.

Performs any necessary ASIS termination activities. This should be invoked once following the last use of other ASIS queries. Parameter values are implementation dependent. The call is ignored if ASIS is already finalized. Subsequent calls to ASIS Environment, Compilation_Unit, and Element queries, are erroneous while the environment Is_Finalized.

Raises ASIS_Failed if the ASIS implementation failed to finalize. Status is likely to be Internal_Error and will not be Not_An_Error.

6.9 function Status

```
function Status return Asis.Errors.Error_Kinds;
```

Returns the Error_Kinds value for the most recent error.

Whenever an error condition is detected, and any ASIS exception is raised, an Asis.Errors.Error_Kinds value and a Diagnosis string are stored. The Status function returns the Asis.Errors.Error_Kinds value for the most recently recorded error.

6.10 function Diagnosis

```
function Diagnosis return Wide_String;
```

Returns a string value describing the most recent error.

Whenever an error condition is detected, and any ASIS exception is raised, an Asis.Errors.Error_Kinds value and a Diagnosis string are stored. The Diagnosis function returns the diagnostic message describing the most recently recorded error. Note that Diagnosis values are implementation dependent and may vary greatly among ASIS implementations.

Will typically return a null string if Status = Not_An_Error.

6.11 procedure Set_Status

```
procedure Set_Status
    (Status    : in Asis.Errors.Error_Kinds := Asis.Errors.Not_An_Error;
     Diagnosis : in Wide_String             := "");
```

Status specifies the new status to be recorded. Diagnosis specifies the new diagnosis to be recorded

Sets (clears, if the defaults are used) the Status and Diagnosis information. Future calls to Status will return this Status (Not_An_Error) and this Diagnosis (a null string).

Raises ASIS_Failed, with a Status of Internal_Error and a Diagnosis of a null string, if the Status parameter is Not_An_Error and the Diagnosis parameter is not a null string.

Section 7: package Asis.Implementation.Permissions

The library package Asis.Implementation.Permissions shall exist. The package shall provide interfaces equivalent to those described in the following subclauses.

l

7.1 function Attributes_Are_Supported

```
function Attributes_Are_Supported return Boolean;
```

Returns True if an implementation supports compilation unit attributes; otherwise, returns False if (all attributes will return Has_Attribute() = False).

7.2 function Implicit_Components_Supported

```
function Implicit_Components_Supported return Boolean;
```

Returns True if the implementation provides elements representing implicit implementation-defined record components, and returns False otherwise.

7.3 function Predefined_Operations_Supported

```
function Predefined_Operations_Supported return Boolean;
```

Returns True if the implementation supports queries of predefined operations, and returns False otherwise.

Section 8: package Asis.Ada_Environments

The library package Asis.Ada_Environments shall exist. The package shall provide interfaces equivalent to those described in the following subclauses.

Asis.Ada_Environments encapsulates a set of queries that map physical Ada compilation and program execution environments to logical ASIS environments.

8.1 function Default_Name

```
function Default_Name return Wide_String;
```

Returns the default context name. If there is no default context name, a null string is returned.

8.2 function Default_Parameters

```
function Default_Parameters return Wide_String;
```

Returns the default context parameters. If there are no default context parameters, a null string is returned.

8.3 procedure Associate

```
procedure Associate
                 (The_Context : in out Asis.Context;
                  Name        : in     Wide_String;
                  Parameters  : in     Wide_String := Default_Parameters);
```

The_Context specifies the Context to associate. Name specifies the name for the Context association. Parameters specifies parameters to use when opening the Context.

Procedure Associate is used to give name and parameter associations to a Context. The Has_Associations query is used to test whether or not a Context has been given name and parameter associations. The Name and Parameters queries are used to examine name and parameter associations.

A Context has at most one set of name/parameter values associated with it at any time. If The_Context is not open then any previous Name and Parameters associations are replaced by this call.

ASIS implementations are encouraged, but not required, to validate the Parameters string immediately. It is recognized that some options cannot be completely validated until the Open is performed. An invalid Parameters value is reported by raising ASIS_Failed with a Status of Parameter_Error.

Raises ASIS_Inappropriate_Context if The_Context is open.

8.4 procedure Open

```
procedure Open (The_Context : in out Asis.Context);
```

The_Context specifies the Context to open.

Opens the ASIS Context using the Context's associated name and parameter values.

Raises ASIS_Inappropriate_Context if The_Context is already open or if it is *unassociated*: does not have associated name and parameter values.

Raises ASIS_Failed if The_Context could not be opened for any reason. The most likely Status values are Name_Error, Use_Error, Data_Error, and Parameter_Error. Other possibilities include Storage_Error and Capacity_Error.

8.5 procedure Close

```
procedure Close (The_Context : in out Asis.Context);
```

The_Context specifies the Context to close.

Closes the ASIS Context. Any previous Context name and parameter associations are retained. This allows the same Context to be re-opened later with the same associations.

All Compilation_Unit and Element values obtained from The_Context become invalid when it is closed. Subsequent calls to ASIS services using such invalid Compilation_Unit or Element values are erroneous. ASIS implementations will attempt to detect such usage and raise ASIS_Failed in response. Applications should be aware that the ability to detect the use of such "dangling references" is implementation specific and not all implementations are able to raise ASIS_Failed at the appropriate points. Thus, applications that attempt to utilize invalid values may exhibit unpredictable behavior.

Raises ASIS_Inappropriate_Context if The_Context is not open.

8.6 procedure Dissociate

```
procedure Dissociate (The_Context : in out Asis.Context);
```

The_Context specifies the Context whose name and parameter associations are to be cleared.

Severs all previous associations for The_Context. A Context that does not have associations (is unassociated) is returned unchanged. The variable The_Context is returned to its unassociated state.

Contexts that have been given Names and Parameters should be Dissociated when they are no longer necessary. Some amount of program storage can be tied up by the stored Name and Parameter strings. This space is only freed when a Context is Dissociated or when ASIS is Finalized.

This operation has no physical affect on any implementor's Ada environment.

Raises ASIS_Inappropriate_Context if The_Context is open.

8.7 function Is_Equal (context)

```
function Is_Equal (Left  : in Asis.Context;
                   Right : in Asis.Context) return Boolean;
```

Left specifies the first Context. Right specifies the second Context.

Returns True if Left and Right designate the same set of associated compilation units, and returns False otherwise. The Context variables may be open or closed.

Unless both Contexts are open, this operation is implemented as a pair of simple string comparisons between the Name and Parameter associations for the two Contexts. If both Contexts are open, this operation acts as a set comparison and returns True if both sets contain the same units (all unit versions are included in the comparison).

8.8 function Is_Identical (context)

```
function Is_Identical (Left  : in Asis.Context;
                       Right : in Asis.Context) return Boolean;
```

Left specifies the first Context. Right specifies the second Context.

Returns True if Left and Right denote the same ASIS context object and that object is open. Returns False otherwise.

8.9 function Exists (context)

```
function Exists (The_Context : in Asis.Context) return Boolean;
```

The_Context specifies a Context to query.

Returns True if The_Context is open or if The_Context designates an Ada environment that can be determined to exist.

Returns False otherwise (in particular, if The_Context is unassociated.

Implementation Permissions

No guarantee is made that The_Context is readable or that an Open operation on The_Context would succeed. The associated parameter value for The_Context may not be fully validated by this simple existence check. It may contain information that can only be verified by an Open.

8.10 function Is_Open

```
function Is_Open (The_Context : in Asis.Context) return Boolean;
```

The_Context specifies the Context to query.

Returns True if The_Context is currently open, and returns False otherwise.

8.11 function Has_Associations

```
function Has_Associations (The_Context : in Asis.Context) return Boolean;
```

The_Context specifies the Context to query.

Returns True if name and parameter values have been associated with The_Context.

Returns False if The_Context is unassociated.

8.12 function Name (context)

```
function Name (The_Context : in Asis.Context) return Wide_String;
```

The_Context specifies the Context to query.

Returns a null string if The_Context is unassociated; otherwise, returns the Name value associated with The_Context.

8.13 function Parameters (context)

```
function Parameters (The_Context : in Asis.Context) return Wide_String;
```

The_Context specifies the Context to query.

Returns a null string if The_Context is unassociated; otherwise, returns the Parameters value associated with The_Context.

8.14 function Debug_Image (context)

```
function Debug_Image (The_Context : in Asis.Context) return Wide_String;
```

The_Context specifies the Context to represent.

Returns a string value containing implementation-defined debugging information associated with The_Context.

The return value uses Asis.Text.Delimiter_Image to separate lines in multi-line results. The return value is not terminated with Asis.Text.Delimiter_Image.

Returns a null string if The_Context is unassociated.

These values are intended for two purposes. They are suitable for inclusion in problem reports sent to the ASIS implementor. They can be presumed to contain information useful when debugging the implementation itself. They are also suitable for use by the ASIS application when printing simple application debugging messages during application development. They are intended to be, to some worthwhile degree, intelligible to the user.

Section 9: package Asis.Ada_Environments.Containers

The library package Asis.Ada_Environments.Containers shall exist. The package shall provide interfaces equivalent to those described in the following subclauses.

If an Ada implementation supports the notion of a program library or “library” as specified in Subclause 10(2) of the Ada Reference Manual, then an ASIS Context value can be mapped onto one or more implementor libraries represented by Containers.

9.1 type Container

The *Container* abstraction is a logical collection of compilation units. For example, one container might hold compilation units which include Ada predefined library units, another container might hold implementation-defined packages. Alternatively, there might be 26 containers, each holding compilation units that begin with their respective letter of the alphabet. The point is that no implementation-independent semantics are associated with a container; it is simply a logical collection.

ASIS implementations shall minimally map the Asis.Context to a list of one ASIS Container whose Name is that of the Asis.Context Name.

```
type Container is private;
Nil_Container : constant Container;

function "=" (Left  : in Container;
              Right : in Container)
              return Boolean is abstract;
```

Nil_Container is the value of a Container that represents no container.

9.2 type Container_List

```
type Container_List is
   array (List_Index range <>) of Container;
```

Type Container_List represents a list of containers.

9.3 function Defining_Containers

```
function Defining_Containers (The_Context : in Asis.Context)
    return Container_List;
```

The_Context specifies the Context to define.

Returns a Container_List value that defines the single environment Context. Each Container will have an Enclosing_Context that Is_Identical to the argument The_Context. The order of Container values in the list is not defined.

Returns a minimal list of length one if the ASIS Ada implementation does not support the concept of a program library. In this case, the Container will have the same name as the given Context.

Raises ASIS_Inappropriate_Context if The_Context is not open.

9.4 function Enclosing_Context (container)

```
function Enclosing_Context (The_Container : in Container)
    return Asis.Context;
```

The_Container specifies the Container to query.

Returns the Context value associated with the Container.

Returns the Context for which the Container value was originally obtained. Container values obtained through the Defining_Containers query will always remember the Context from which they were defined.

Because Context is limited private, this function is only intended to be used to supply a Context parameter for other queries.

Raises ASIS_Inappropriate_Container if the Container is a Nil_Container.

9.5 function Library_Unit_Declarations (container)

```
function Library_Unit_Declarations (The_Container : in Container)
                                   return Asis.Compilation_Unit_List;
```

The_Container specifies the Container to query.

Returns a list of all library_unit_declaration and library_unit_renaming_declaration elements contained in the Container. Individual units will appear only once in an order that is not defined.

A Nil_Compilation_Unit_List is returned if there are no declarations of library units within the Container.

This query will never return a unit with A_Configuration_Compilation or a Nonexistent unit kind. It will never return a unit with A_Procedure_Body or A_Function_Body unit kind even though the unit is interpreted as both the declaration and body of a library procedure or library function. (Reference Manual 10.1.4(4).

All units in the result will have an Enclosing_Container value that Is_Identical to the Container.

Raises ASIS_Inappropriate_Context if the Enclosing_Context(Container) is not open.

9.6 function Compilation_Unit_Bodies (container)

```
function Compilation_Unit_Bodies (The_Container : in Container)
                                 return Asis.Compilation_Unit_List;
```

The_Container specifies the Container to query.

Returns a list of all library_unit_body and subunit elements contained in the Container. Individual units will appear only once in an order that is not defined.

A Nil_Compilation_Unit_List is returned if there are no bodies within the Container.

This query will never return a unit with A_Configuration_Compilation or a nonexistent unit kind.

All units in the result will have an Enclosing_Container value that Is_Identical to the Container.

Raises ASIS_Inappropriate_Context if the Enclosing_Context(Container) is not open.

9.7 function Compilation_Units (container)

```
function Compilation_Units (The_Container : in Container)
                           return Asis.Compilation_Unit_List;
```

The_Container specifies the Container to query.

Returns a list of all compilation units contained in the Container. Individual units will appear only once in an order that is not defined.

A Nil_Compilation_Unit_List is returned if there are no units within the Container.

This query will never return a unit with A_Configuration_Compilation or a nonexistent unit kind.

All units in the result will have an Enclosing_Container value that Is_Identical to the Container.

Raises ASIS_Inappropriate_Context if the Enclosing_Context(Container) is not open.

9.8 function Is_Equal (containers)

```
function Is_Equal (Left  : in Container;
                   Right : in Container) return Boolean;
```

Left specifies the first Container. Right specifies the second Container.

Returns True if Left and Right designate Container values that contain the same set of compilation units, and returns False otherwise. The Container values may have been defined from different Context values.

9.9 function Is_Identical (containers)

```
function Is_Identical (Left  : in Container;
                       Right : in Container) return Boolean;
```

Left specifies the first Container. Right specifies the second Container.

Returns True if Is_Equal(Left, Right) and the Container values have been defined from Is_Equal Context values, and returns False otherwise.

9.10 function Name (container)

```
function Name (The_Container : in Container) return Wide_String;
```

The_Container specifies the Container to name.

Returns the Name value associated with the Container.

Returns a null string if the Container is a Nil_Container.

Section 10: package Asis.Compilation_Units

The library package Asis.Compilation_Units shall exist. The package shall provide interfaces equivalent to those described in the following subclauses.

Asis.Compilation_Units encapsulates a set of queries that implement the ASIS Compilation_Unit abstraction.

More than one compilation unit may be manipulated at one time. (The exact number is subject to implementation specific limitations.)

A specific Compilation_Unit value is valid (usable) for as long as the ASIS Context variable used to create it remains open. Once an ASIS Context is closed, all associated Compilation_Unit values become invalid. It is erroneous to use an invalid Compilation_Unit value.

10.1 function Unit_Kind

```
function Unit_Kind (Compilation_Unit : in Asis.Compilation_Unit)
           return Asis.Unit_Kinds;
```

Compilation_Unit specifies the compilation unit to query.

Returns the Unit_Kinds value of the compilation unit. Returns Not_A_Unit for a Nil_Compilation_Unit.

Compilation_Unit expects any kind of unit.

Returns An_Unknown_Unit for any compilation unit that exists, but that does not have semantic element information available through ASIS.

Returns a nonexistent kind for units that have name-only entries in the environment Context. Such entries may exist for names because:

- They represent an illegal compilation unit added to the environment.
- They are referenced by some existing unit, but the program text for the referenced unit has never been supplied, compiled, or otherwise inserted into the environment.
- They represent a separate subunit that has never been supplied, compiled, or otherwise inserted into the environment.
- The unit may have existed at one time but the semantic information is no longer available. It may be inconsistent, have been removed by some user or Ada environment operations, or simply have been lost as the result of some sort of failure.

10.2 function Unit_Class

```
function Unit_Class (Compilation_Unit : in Asis.Compilation_Unit)
           return Asis.Unit_Classes;
```

Compilation_Unit specifies the compilation unit to query.

Returns the Unit_Classes value of the compilation unit. Returns Not_A_Class for a Nil_Compilation_Unit.

Compilation_Unit expects any kind of unit.

10.3 function Unit_Origin

```
function Unit_Origin (Compilation_Unit : in Asis.Compilation_Unit)
           return Asis.Unit_Origins;
```

Compilation_Unit specifies the compilation unit to query.

Returns the Unit_Origins value of the unit. Returns Not_An_Origin for a compilation_unit whose Unit_Kind is Not_A_Unit, A_Nonexistent_Declaration, A_Nonexistent_Body, or possibly An_Unknown_Unit.

Compilation_Unit expects any kind of unit.

Implementation Permissions

ASIS queries on the private part and bodies of Compilation_Units whose Unit_Origin is other than An_Application_Unit may have limitations that are implementation-defined. Operations of package Asis.Text on Compilation_Units whose Unit_Origin is other than An_Application_Unit may have limitations that are implementation-defined.

10.4 function Enclosing_Context (unit)

```
function Enclosing_Context (Compilation_Unit : in Asis.Compilation_Unit)
           return Asis.Context;
```

Compilation_Unit specifies the unit whose Context is required.

Returns the Context containing the compilation unit.

Compilation units always remember the ASIS Context and Container from which they were obtained.

This function can be used to eliminate the need to make the original Context visible at the place of each call where a Context parameter is required.

Two Compilation_Unit values, that represent the same physical compilation units (same Ada implementor Context implementation unit value) will test as Is_Equal, but not Is_Identical, if they were obtained from different open ASIS Context variables.

Raises ASIS_Inappropriate_Compilation_Unit with a Status of Value_Error if the unit is a Nil_Compilation_Unit.

10.5 function Enclosing_Container

```
function Enclosing_Container (Compilation_Unit : in Asis.Compilation_Unit)
           return Asis.Ada_Environments.Containers.Container;
```

Compilation_Unit specifies the unit whose Container is required.

Returns the Container of the Context containing the compilation unit. Compilation units always remember the ASIS Context and Container from which they were obtained.

Raises ASIS_Inappropriate_Compilation_Unit with a Status of Value_Error if the unit is a Nil_Compilation_Unit.

10.6 function Library_Unit_Declaration

```
function Library_Unit_Declaration (Name        : in Wide_String;
                                   The_Context : in Asis.Context)
                                    return Asis.Compilation_Unit;
```

Name specifies the defining program unit name. The_Context specifies a program Context environment.

Returns the library_unit_declaration or library_unit_renaming_declaration with the specified Name, contained in The_Context.

This query will never return a unit with A_Configuration_Compilation or a nonexistent unit kind. It will never return a unit with A_Procedure_Body or A_Function_Body unit kind even though the unit is interpreted as both the declaration and body of a library procedure or library function. (Ada Standard 10.1.4(4).)

A Nil_Compilation_Unit is returned if no such declaration exists.

Any non-Nil result will have an Enclosing_Context value that Is_Identical to the Context. Never returns a unit with a nonexistent unit kind.

Raises ASIS_Inappropriate_Context with a Status of Value_Error if The_Context is not open.

10.7 function Compilation_Unit_Body

```
function Compilation_Unit_Body (Name        : in Wide_String;
                                The_Context : in Asis.Context)
                                return Asis.Compilation_Unit;
```

Name specifies the defining_program_unit_name. The_Context specifies a program Context environment.

Returns the library_unit_body or subunit with the name, contained in the library.

A Nil_Compilation_Unit is returned if no such body exists.

Any non-Nil result will have an Enclosing_Context value that Is_Identical to The_Context. Never returns a unit with a nonexistent unit kind.

Raises ASIS_Inappropriate_Context with a Status of Value_Error if The_Context is not open.

10.8 function Library_Unit_Declarations (context)

```
function Library_Unit_Declarations (The_Context : in Asis.Context)
                                    return Asis.Compilation_Unit_List;
```

The_Context specifies a program Context environment.

Returns a list of all library_unit_declaration and library_unit_renaming_declaration elements contained in The_Context. Individual units will appear only once in an order that is not defined.

A Nil_Compilation_Unit_List is returned if there are no declarations of library units within The_Context.

This query will never return a unit with A_Configuration_Compilation or a nonexistent unit kind. It will never return a unit with A_Procedure_Body or A_Function_Body unit kind even though the unit is interpreted as both the declaration and body of a library procedure or library function. (Ada Standard 10.1.4(4).)

All units in the result will have an Enclosing_Context value that Is_Identical to The_Context.

Raises ASIS_Inappropriate_Context with a Status of Value_Error if The_Context is not open.

10.9 function Compilation_Unit_Bodies (context)

```
function Compilation_Unit_Bodies (The_Context : in Asis.Context)
                                  return Asis.Compilation_Unit_List;
```

The_Context specifies a program Context environment.

Returns a list of all library_unit_body and subunit elements contained in The_Context. Individual units will appear only once in an order that is not defined.

A Nil_Compilation_Unit_List is returned if there are no bodies within The_Context.

This query will never return a unit with A_Configuration_Compilation or a nonexistent unit kind.

All units in the result will have an Enclosing_Context value that Is_Identical to The_Context.

Raises ASIS_Inappropriate_Context with a Status of Value_Error if The_Context is not open.

10.10 function Compilation_Units (context)

```
function Compilation_Units (The_Context : in Asis.Context)
                           return Asis.Compilation_Unit_List;
```

The_Context specifies a program Context environment.

Returns a list of all compilation units contained in The_Context. Individual units will appear only once in an order that is not defined.

A Nil_Compilation_Unit_List is returned if there are no units within The_Context.

This query will never return a unit with A_Configuration_Compilation or a nonexistent unit kind.

All units in the result will have an Enclosing_Context value that Is_Identical to The_Context.

Raises ASIS_Inappropriate_Context with a Status of Value_Error if The_Context is not open.

10.11 function Corresponding_Children

```
function Corresponding_Children (Library_Unit : in Asis.Compilation_Unit)
                  return Asis.Compilation_Unit_List;
function Corresponding_Children (Library_Unit : in Asis.Compilation_Unit;
                                 The_Context  : in Asis.Context)
                  return Asis.Compilation_Unit_List;
```

Library_Unit specifies the library unit whose children are desired. The_Context specifies a program Context environment.

Returns a list of the child units for the given parent library unit.

Both the declaration and body (if any) of each child unit are returned. Descendants beyond immediate children (i.e., children of children) are not returned by this query.

Use the compilation unit relationship queries with a Relation_Kinds of Descendants to create a list of children, children of children, and so on.

Returns a Nil_Compilation_Unit_List for all library unit arguments that do not have any child units contained in The_Context.

If The_Context specifies a context that is different from the Enclosing_Context(Library_Unit), returns a list of the child units for the parent library unit that Is_Equal to Library_Unit in The_Context. If no such parent library unit exists in The_Context, returns a Nil_Compilation_Unit_List.

These two function calls will always produce identical results:

```
Units := Corresponding_Children (Unit);
Units := Corresponding_Children (Unit, Enclosing_Context (Unit));
```

The Enclosing_Context for any non-Nil result will always be Is_Identical to The_Context, regardless of the Enclosing_Context value for the Library_Unit argument. This query is one means of obtaining (Is_Equal) child units from separate ASIS Context values whose underlying implementations overlap.

Library_Unit expects a unit that has one of the following Unit_Kinds:

A_Package
A_Generic_Package
A_Package_Instance

Raises ASIS_Inappropriate_Compilation_Unit with a Status of Value_Error for any unit that does not have one of these expected kinds.

Raises ASIS_Inappropriate_Context with a Status of Value_Error if The_Context is provided and is not open.

Returns a list of units that each have one of the following Unit_Kinds:

A_Procedure
A_Function
A_Package
A_Generic_Procedure
A_Generic_Function
A_Generic_Package
A_Procedure_Instance
A_Function_Instance
A_Package_Instance
A_Procedure_Renaming
A_Function_Renaming
A_Package_Renaming
A_Generic_Procedure_Renaming
A_Generic_Function_Renaming
A_Generic_Package_Renaming
A_Procedure_Body
A_Function_Body
A_Package_Body
An_Unknown_Unit

If the declaration of a child is inconsistent with the argument of the query, neither the declaration nor the body is returned. If the declaration of a child is consistent with the argument, but the body is not, the declaration is returned, and for the body, the result of the Corresponding_Body query applied to the declaration is returned.

10.12 function Corresponding_Parent_Declaration

```
function Corresponding_Parent_Declaration
           (Library_Unit : in Asis.Compilation_Unit)
           return Asis.Compilation_Unit;
```

```
function Corresponding_Parent_Declaration
            (Library_Unit : in Asis.Compilation_Unit;
             The_Context  : in Asis.Context)
            return Asis.Compilation_Unit;
```

Library_Unit specifies the unit whose parent is desired. The_Context specifies a program Context environment.

Returns the parent unit of the given library unit.

Returns a Nil_Compilation_Unit if the Library_Unit argument represents package Standard. Root Library_Unit arguments return the package Standard.

Returns A_Nonexistent_Declaration when the Library_Unit has a parent_unit_name denoted in the defining_program_unit_name but the parent unit is not contained in The_Context.

If The_Context specifies a context that is different from the Enclosing_Context(Library_Unit), returns the parent library unit for the child unit in The_Context that Is_Equal to Library_Item. If no such child unit exists in The_Context, returns a Nil_Compilation_Unit.

These two function calls will always produce identical results:

```
Unit := Corresponding_Parent_Declaration (Unit);
Unit := Corresponding_Parent_Declaration (Unit, Enclosing_Context
(Unit));
```

The Enclosing_Context for any non-Nil result will always be Is_Identical to The_Context, regardless of the Enclosing_Context value for the Library_Unit argument. This query is one means of obtaining (Is_Equal) parent units from separate ASIS Context values whose underlying implementations overlap.

Library_Unit expects a unit that has one of the following Unit_Kinds:

A_Procedure
A_Function
A_Package
A_Generic_Procedure
A_Generic_Function
A_Generic_Package
A_Procedure_Instance
A_Function_Instance
A_Package_Instance
A_Procedure_Renaming
A_Function_Renaming
A_Package_Renaming
A_Generic_Procedure_Renaming
A_Generic_Function_Renaming
A_Generic_Package_Renaming
A_Procedure_Body
A_Function_Body
A_Package_Body

Raises ASIS_Inappropriate_Compilation_Unit with a Status of Value_Error for any unit that does not have one of these expected kinds.

Raises ASIS_Inappropriate_Context with a Status of Value_Error if The_Context is provided and is not open.

Returns a unit with one of the following Unit_Kinds:

Not_A_Unit

A_Package
A_Generic_Package
A_Package_Instance
A_Nonexistent_Declaration
An_Unknown_Unit

If a parent is inconsistent with a child passed as the argument, A_Nonexistent_Declaration shall be returned.

10.13 function Corresponding_Declaration (unit)

```
function Corresponding_Declaration
            (Library_Item : in Asis.Compilation_Unit)
            return Asis.Compilation_Unit;

function Corresponding_Declaration
            (Library_Item : in Asis.Compilation_Unit;
             The_Context  : in Asis.Context)
            return Asis.Compilation_Unit;
```

Library_Item specifies the library_item whose declaration is desired. The_Context specifies a program Context environment.

Returns the corresponding library_unit_declaration, if any, for the library_unit_body. The corresponding library unit is the unit upon which the library_unit_body depends semantically.

Returns a unit that Is_Equal to the argument if:

- the argument is a library_unit_declaration, a library_unit_renaming_declaration, or a subunit.
- the argument is A_Nonexistent_Declaration or A_Nonexistent_Body.

Returns a Nil_Compilation_Unit for library_unit_body arguments that do not have a corresponding library unit contained in The_Context.

If The_Context specifies a context that is different from the Enclosing_Context(Library_Unit), returns the corresponding library unit declaration for the library unit body in The_Context that Is_Equal to Library_Item. If no such library unit body exists in The_Context, returns a Nil_Compilation_Unit.

These two function calls will always produce identical results:

```
Unit := Corresponding_Declaration (Unit);
Unit := Corresponding_Declaration (Unit, Enclosing_Context (Unit));
```

The Enclosing_Context for any non-Nil result will always be Is_Identical to The_Context, regardless of the Enclosing_Context value for the Library_Item argument. This query is one means of obtaining corresponding (Is_Equal) units from separate ASIS Context values whose underlying implementations overlap.

Library_Item expects a unit of any Unit_Kinds except Not_A_Unit.

Raises ASIS_Inappropriate_Compilation_Unit with a Status of Value_Error if Library_Item has Unit_Kinds Not_A_Unit.

Raises ASIS_Inappropriate_Context with a Status of Value_Error if The_Context is provided and is not open.

Returns a unit that has one of the following Unit_Kinds::

A_Procedure
A_Function
A_Package
A_Generic_Procedure

A_Generic_Function
A_Generic_Package
A_Procedure_Instance
A_Function_Instance
A_Package_Instance
A_Procedure_Renaming
A_Function_Renaming
A_Package_Renaming
A_Generic_Procedure_Renaming
A_Generic_Function_Renaming
A_Generic_Package_Renaming
A_Procedure_Body_Subunit
A_Function_Body_Subunit
A_Package_Body_Subunit
A_Task_Body_Subunit
A_Protected_Body_Subunit
A_Nonexistent_Declaration
A_Nonexistent_Body

If the declaration of an argument Element is inconsistent with the argument, A_Nonexistent_Declaration shall be returned. (For a unit A_Procedure_Body or A_Function_Body kind, the solution may be in any case, to return Nil_Compilation_Unit if the unit is of A_Public_Declaration_And_Body kind.)

Implementation Permissions

The handling of An_Unknown_Unit is implementation specific. The expected use for An_Unknown_Unit is to hide proprietary implementation details contained within unit bodies. In these cases, it should be possible to obtain an appropriate library_unit_declaration when starting with An_Unknown_Unit. Some implementors may choose to simply return the An_Unknown_Unit argument in all cases.

10.14 function Corresponding_Body (unit)

```
function Corresponding_Body
            (Library_Item : in Asis.Compilation_Unit)
            return Asis.Compilation_Unit;

function Corresponding_Body
            (Library_Item : in Asis.Compilation_Unit;
             The_Context  : in Asis.Context)
            return Asis.Compilation_Unit;
```

Library_Item specifies the library_item whose body is desired. The_Context specifies a program Context environment.

Returns the corresponding library_unit_body, if any, for the library_unit_declaration. The corresponding library_unit_body is the unit that depends semantically on the library_unit_declaration.

Returns a unit that Is_Equal to the argument if:

- the argument is an instance of a library_unit_declaration, a library_unit_body, a library_unit_renaming_declaration, or a subunit.
- the argument is A_Nonexistent_Declaration or A_Nonexistent_Body.

Returns a Nil_Compilation_Unit for library_unit_declaration arguments that do not have a corresponding library_unit_body contained in The_Context.

If The_Context specifies a context that is different from the Enclosing_Context(Library_Unit), returns the corresponding library unit body for the library unit in The_Context that Is_Equal to Library_Item. If no such library unit exists in The_Context, returns a Nil_Compilation_Unit.

These two function calls will always produce identical results:

```
Unit := Corresponding_Body (Unit);
Unit := Corresponding_Body (Unit, Enclosing_Context (Unit));
```

The Enclosing_Context for any non-Nil result will always be Is_Identical to The_Context, regardless of the Enclosing_Context value for the Library_Item argument. This query is one means of obtaining corresponding (Is_Equal) units from separate ASIS Context values whose underlying implementations overlap.

Library_Item expects a unit of any Unit_Kinds except Not_A_Unit.

Raises ASIS_Inappropriate_Compilation_Unit with a Status of Value_Error if Library_Item has Unit_Kinds Not_A_Unit.

Raises ASIS_Inappropriate_Context with a Status of Value_Error if The_Context is provided and is not open.

Returns a unit that has one of the following Unit_Kinds:

A_Procedure_Body
A_Function_Body
A_Package_Body
A_Procedure_Instance
A_Function_Instance
A_Package_Instance
A_Procedure_Renaming
A_Function_Renaming
A_Package_Renaming
A_Generic_Procedure_Renaming
A_Generic_Function_Renaming
A_Generic_Package_Renaming
A_Procedure_Body_Subunit
A_Function_Body_Subunit
A_Package_Body_Subunit
A_Task_Body_Subunit
A_Protected_Body_Subunit
A_Nonexistent_Declaration
A_Nonexistent_Body

If the argument Element requires a body to be presented to make up a complete partition containing this Element, but The_Context does not contain the corresponding body, or the body contained in The_Context is inconsistent with the argument Element, A_Nonexistent_Body shall be returned.

Implementation Permissions

The handling of An_Unknown_Unit is implementation specific. The expected use for An_Unknown_Unit is to hide proprietary implementation details contained within unit bodies. In some cases, it could be possible to obtain an appropriate library_unit_body when starting with An_Unknown_Unit. Some implementors may choose to simply return the An_Unknown_Unit argument in all cases.

10.15 function Is_Nil (unit)

```
function Is_Nil (Right : in Asis.Compilation_Unit)
                 return Boolean;
```

Right specifies the unit to test.

Returns True if the compilation_unit is a Nil_Compilation_Unit, and returns False otherwise.

10.16 function Is_Nil (unit list)

```
function Is_Nil (Right : in Asis.Compilation_Unit_List)
                return Boolean;
```

Right specifies the unit list to test.

Returns True if the compilation_unit list has a length of zero, and returns False otherwise.

10.17 function Is_Equal (unit)

```
function Is_Equal (Left  : in Asis.Compilation_Unit;
                   Right : in Asis.Compilation_Unit) return Boolean;
```

Left specifies the first unit to compare. Right specifies the second unit to compare.

Returns True if Left and Right represent the same physical compilation unit or if both are Nil_Compilation_Unit values, and returns False otherwise. The two units may or may not be from the same ASIS Context variable. (The *same physical compilation unit* has the same version, as defined by Ada Standard E.3(5) and the same program text.)

Two nonexistent units are Is_Equal if they have the same Name and Unit_Kind.

10.18 function Is_Identical

```
function Is_Identical (Left  : in Asis.Compilation_Unit;
                       Right : in Asis.Compilation_Unit) return Boolean;
```

Left specifies the first unit to compare. Right specifies the second unit to compare.

Returns True if Left and Right represent the same physical compilation unit from the same open ASIS Context variable or if both are Nil_Compilation_Unit values, and returns False otherwise.

Two nonexistent units are Is_Identical if they have the same Unique_Name and the same Enclosing_Context.

10.19 function Unit_Full_Name

```
function Unit_Full_Name (Compilation_Unit : in Asis.Compilation_Unit)
                         return Wide_String;
```

```
Compilation_Unit specifies the unit whose name is desired.
```

Returns the string image of the fully expanded Ada name of the given compilation unit. This may be a simple name ("A") of a root library unit, or an expanded name ("A.B") of a subunit or non-root child unit. An expanded name shall contain the full parent_unit_name as its prefix.

Returns a null string only if A_Configuration_Compilation or a Nil_Compilation_Unit is given.

The case of names returned by this query may vary between implementations. Implementors are encouraged, but not required, to return names in the same case as was used in the original compilation text.

Compilation_Unit expects any kind of unit.

10.20 function Unique_Name

```
function Unique_Name
     (Compilation_Unit : in Asis.Compilation_Unit) return Wide_String;
```

Compilation_Unit specifies the unit. whose name is desired.

Returns a string that uniquely identifies the given compilation unit within the underlying Ada Context implementation. The result may vary depending on the ASIS implementation. The unique name may include the name and parameters of the Context, file system paths, library files, version numbers, kind, or any other information that an implementation may need to uniquely identify the compilation unit.

Returns a null string only if a Nil_Compilation_Unit is given.

Compilation_Unit expects any kind of unit.

10.21 function Exists (unit)

```
function Exists (Compilation_Unit : in Asis.Compilation_Unit)
        return Boolean;
```

Compilation_Unit specifies the unit to test.

Returns False for any unit with Not_A_Unit or nonexistent kind. Returns True for all other unit kinds.

Compilation_Unit expects any kind of unit.

10.22 function Can_Be_Main_Program

```
function Can_Be_Main_Program (Compilation_Unit : in Asis.Compilation_Unit)
                             return Boolean;
```

Compilation_Unit specifies the unit to test.

Returns True if the Compilation_Unit exists and is a subprogram library_unit_declaration, library_unit_renaming_declaration, or library_unit_body that can be used as a main subprogram. See Ada Standard 10.2(7).

Returns False otherwise.

Results of this function may vary according to the requirements an Ada implementation may impose on a main subprogram.

Compilation_Unit expects any kind of unit.

10.23 function Is_Body_Required

```
function Is_Body_Required (Compilation_Unit : in Asis.Compilation_Unit)
                             return Boolean;
```

Compilation_Unit specifies the unit to test.

Returns True if the Compilation_Unit exists and requires a body to be present to make up a complete partition; otherwise returns False.

Compilation_Unit expects any kind of unit.

10.24 function Text_Name

```
function Text_Name (Compilation_Unit : in Asis.Compilation_Unit)
         return Wide_String;
```

Compilation_Unit specifies the unit whose text name is desired.

Returns the name of the text, or other structure, that was the source of the compilation that resulted in this Compilation_Unit. Returns a null string if the unit has a Nil or nonexistent kind, or if the text name is not available for any reason.

Ada has no concept of source or text file. Text_Name availability is a required feature of ASIS. Results of this function may vary among implementations.

Compilation_Unit expects any kind of unit.

10.25 function Text_Form

```
function Text_Form (Compilation_Unit : in Asis.Compilation_Unit)
         return Wide_String;
```

Compilation_Unit specifies the unit whose text form is desired.

Returns the Form parameter (as for Ada.Text_Io.Open) for the text, or other structure, that was the source of the compilation that resulted in this Compilation_Unit. Returns a null string if the unit has a Nil or nonexistent kind, if the text was created with an empty Form parameter, or if the text Form parameter value is not available for any reason.

Ada has no concept of source or text file. Text_Form availability is a required feature of ASIS. Results of this function may vary among implementations.

Compilation_Unit expects any kind of unit.

10.26 function Object_Name

```
function Object_Name (Compilation_Unit : in Asis.Compilation_Unit)
         return Wide_String;
```

Compilation_Unit specifies the unit whose object name is desired.

Returns the name of the object, or other structure, that contains the binary result of the compilation for this Compilation_Unit. Returns a null string if the unit has a Nil or nonexistent kind, or if the object name is not available for any reason.

Compilation_Unit expects any kind of unit.

10.27 function Object_Form

```
function Object_Form (Compilation_Unit : in Asis.Compilation_Unit)
         return Wide_String;
```

Compilation_Unit specifies the unit whose object form is desired.

Returns the Form parameter (as for Ada.Text_Io.Open) for the object, or other structure, that was the machine-code result of the compilation of this Compilation_Unit. Returns a null string if the unit has a Nil or nonexistent kind, if the object was created with an empty Form parameter, or if the object Form parameter value is not available for any reason.

Compilation_Unit expects any kind of unit.

10.28 function Compilation_Command_Line_Options

```
function Compilation_Command_Line_Options
         (Compilation_Unit : in Asis.Compilation_Unit)
          return Wide_String;
```

Compilation_Unit specifies the unit to query.

Returns the command line options used to compile the Compilation_Unit. Returns null string if the unit has a Nil or nonexistent unit kind, or if the command line options are not available for any reason.

Compilation_Unit expects any kind of unit.

10.29 function Has_Attribute

```
function Has_Attribute (Compilation_Unit : in Asis.Compilation_Unit;
                        Attribute        : in Wide_String) return Boolean;
```

Compilation_Unit specifies the unit to query. Attribute specifies the name of the attribute to query.

Returns True if the compilation unit has the given attribute.

Returns False if the unit is a Nil_Compilation_Unit argument, the unit does not have the given Attribute, or the implementation does not support attributes.]}

Compilation_Unit expects any kind of unit.

Results of this query may vary across ASIS implementations.

10.30 function Attribute_Value_Delimiter

```
function Attribute_Value_Delimiter return Wide_String;
```

Returns the string used as a delimiter separating individual values within the string Attribute_Values of a compilation unit.

Results of this query may vary across ASIS implementations. The result can be a null string for implementations that do not support attributes, or that do not support more than one attribute.

10.31 function Attribute_Values

```
function Attribute_Values
         (Compilation_Unit : in Asis.Compilation_Unit;
          Attribute        : in Wide_String)
          return Wide_String;
```

Compilation_Unit specifies the unit to query. Attribute specifies the name of the attribute to query.

Returns a string containing zero or more images of values that are associated with the given attribute. When more than one value is returned, the Attribute_Value_Delimiter string is used to separate the individual values. Returns a null string if the unit is a Nil_Compilation_Unit argument, the unit has no values for this Attribute, or the implementation does not support attributes.

Compilation_Unit expects any kind of unit.

Results of this query may vary across ASIS implementations.

10.32 function Subunits

```
function Subunits (Parent_Body : in Asis.Compilation_Unit)
         return Asis.Compilation_Unit_List;

function Subunits (Parent_Body : in Asis.Compilation_Unit;
                   The_Context : in Asis.Context)
         return Asis.Compilation_Unit_List;
```

Parent_Body specifies the parent unit to query. The_Context specifies the program Context to use for context.

Returns a complete list of subunit values, with one value for each body stub that appears in the given Parent_Body. Returns a Nil_Compilation_Unit_List if the parent unit does not contain any body stubs. Every unit in the result will have an Enclosing_Context that Is_Identical to The_Context.

These two function calls will always produce identical results:

```
SUnits := Subunits (PUnit);
SUnits := Subunits (PUnit, Enclosing_Context (PUnit));
```

The result may include unit values with a nonexistent unit kind. It includes values for subunits that exist in The_Context as well as values for subunits that do not exist, but whose name can be deduced from the body stub and the name of the parent unit. These nonexistent units are known to be library_unit_body elements so their unit kind is A_Nonexistent_Body.

Subunit lists are also available through the Semantic_Dependence_Order query using the Family relation.

Raises ASIS_Inappropriate_Compilation_Unit with a Status of Value_Error if the unit Parent_Body is a Nil_Compilation_Unit.

Raises ASIS_Inappropriate_Context with a Status of Value_Error if The_Context is provided and is not open.

If a subunit is absent or if it is inconsistent with the argument Element, A_Nonexistent_Body shall be returned for it.

Returns a list of units that each have one of the followingUnit_Kinds:

A_Nonexistent_Body
A_Procedure_Body_Subunit
A_Function_Body_Subunit
A_Package_Body_Subunit
A_Task_Body_Subunit
A_Protected_Body_Subunit

10.33 function Corresponding_Subunit_Parent_Body

```
function Corresponding_Subunit_Parent_Body
           (Subunit : in Asis.Compilation_Unit)
            return Asis.Compilation_Unit;

function Corresponding_Subunit_Parent_Body
           (Subunit     : in Asis.Compilation_Unit;
            The_Context : in Asis.Context)
            return Asis.Compilation_Unit;
```

Subunit specifies the subunit to query. The_Context specifies the program Context to use for context.

Returns the Compilation_Unit containing the body stub of the given Subunit. Returns a Nil_Compilation_Unit if the subunit parent is not contained in The_Context. Any non-Nil result will have an Enclosing_Context value that Is_Identical to The_Context.

These two function calls will always produce identical results:

```
PUnit := Corresponding_Subunit_Parent_Body (SUnit);
PUnit := Corresponding_Subunit_Parent_Body (SUnit,
                                    Enclosing_Context (SUnit));
```

Subunit expects a unit that has one of the following Unit_Kinds:

A_Procedure_Body_Subunit
A_Function_Body_Subunit
A_Package_Body_Subunit
A_Task_Body_Subunit
A_Protected_Body_Subunit

Raises ASIS_Inappropriate_Compilation_Unit with a Status of Value_Error for any unit that does not have one of these expected kinds.

Raises ASIS_Inappropriate_Context with a Status of Value_Error if The_Context is provided and is not open.

Returns a unit that has one of the following Unit_Kinds:

A_Procedure_Body
A_Function_Body
A_Package_Body
A_Procedure_Body_Subunit
A_Function_Body_Subunit
A_Package_Body_Subunit
A_Task_Body_Subunit
A_Protected_Body_Subunit

If the corresponding body does not exist in The_Context, or if it exists, but is inconsistent with the argument Element, then A_Nonexistent_Body shall be returned. To locate the parent of a subunit that is not itself a subunit, repeatedly call Corresponding_Subunit_Parent_Body until a unit that is not a subunit is returned.

10.34 function Debug_Image (unit)

```
function Debug_Image (Compilation_Unit : in Asis.Compilation_Unit)
         return Wide_String;
```

Compilation_Unit specifies a unit to convert.

Returns a string value containing implementation-defined debug information associated with the compilation unit.

The return value uses Asis.Text.Delimiter_Image to separate the lines of multi-line results. The return value does not end with Asis.Text.Delimiter_Image.

These values are intended for two purposes. They are suitable for inclusion in problem reports sent to the ASIS implementor. They can be presumed to contain information useful when debugging the implementation itself. They are also suitable for use by the ASIS application when printing simple application debugging messages during application development. They are intended to be, to some worthwhile degree, intelligible to the user.

Section 11: package Asis.Compilation_Units.Times

The library package Asis.Compilation_Units.Times shall exist. The package shall provide interfaces equivalent to those described in the following subclauses.

Asis.Compilation_Units.Times encapsulates the time related functions used within ASIS.

11.1 type Time

ASIS uses the predefined Ada.Calendar.Time and Standard.Duration for time-related values. The constant Nil_ASIS_Time is defined to support time queries where a time is unavailable/unknown.

```
Nil_ASIS_Time : constant Ada.Calendar.Time :=
       Ada.Calendar.Time_Of (Year    => 1901,
                             Month   => 1,
                             Day     => 1,
                             Seconds => 0.0);
```

11.2 function Time_Of_Last_Update

```
function Time_Of_Last_Update (Compilation_Unit : in Asis.Compilation_Unit)
                             return Ada.Calendar.Time;
```

Compilation_Unit specifies the unit to query.

Returns the time that this physical compilation unit was most recently updated in its implementor's Ada Environment. This will often be the time of its last compilation. The exact significance of the result is implementation specific. Returns Nil_ASIS_Time if the unit has a Nil or nonexistent unit kind, or if the time of last update is not available, or not meaningful, for any reason.

Compilation_Unit expects any kind of unit.

11.3 function Compilation_CPU_Duration

```
function Compilation_CPU_Duration (Compilation_Unit : in Asis.Compilation_Unit)
                                  return Standard.Duration;
```

Compilation_Unit specifies the unit to query.

Returns the Central Processing Unit (CPU) duration used to compile the physical compilation unit associated with the Compilation_Unit argument. The exact significance, or accuracy, of the result is implementation specific. Returns a duration of 0.0 if the unit has a Nil or nonexistent unit kind, or if the CPU duration for the last compilation is not available for any reason. Returns a duration of 86_400.0 if the CPU duration for the last compilation is greater than 1 day.

Compilation_Unit expects any kind of unit.

11.4 function Attribute_Time

```
function Attribute_Time
            (Compilation_Unit : in Asis.Compilation_Unit;
             Attribute        : in Wide_String)
            return Ada.Calendar.Time;
```

Compilation_Unit specifies the unit to query. Attribute specifies the name of the attribute to query.

Returns the Time value associated with the given attribute. Returns Nil_ASIS_Time if the argument is a Nil_Compilation_Unit, the unit does not have the given Attribute, or the implementation does not record times for attributes.

Compilation_Unit expects any kind of unit.

Results of this query may vary across ASIS implementations.

Section 12: package Asis.Compilation_Units.Relations

The library package Asis.Compilation_Units.Relations shall exist. The package shall provide interfaces equivalent to those described in the following subclauses.

Asis.Compilation_Units.Relations encapsulates semantic relationship concepts used in ASIS.

12.1 type Relationship

Relationship queries provide references to compilation units that are related, in some specific fashion, to one or more given compilation units. Compilation units located by these queries are returned as a set of ordered lists.

```
type Relationship (Consistent_Length   : Asis.ASIS_Natural;
                   Inconsistent_Length : Asis.ASIS_Natural;
                   Missing_Length      : Asis.ASIS_Natural;
                   Circular_Length     : Asis.ASIS_Natural) is
   record
      Consistent   : Asis.Compilation_Unit_List (1 .. Consistent_Length);
      Inconsistent : Asis.Compilation_Unit_List (1 .. Inconsistent_Length);
      Missing      : Asis.Compilation_Unit_List (1 .. Missing_Length);
      Circular     : Asis.Compilation_Unit_List (1 .. Circular_Length);
   end record;
```

The following describes the semantics of the unit lists returned by the queries Semantic_Dependence_Order and Elaboration_Order:

Each query returns a set of four lists. Every unit returned will have the same Enclosing_Context. The lists are:

- **Consistent**: A list of consistent ordered units.
- **Inconsistent**: A list of units that are inconsistent with one or more units on which they semantically depend.
- **Missing**: A list of units that have missing (nonexistent) related units.
- **Circular**: A list of circular semantic dependencies between units.

These lists are further described as:

- Consistent units list:

  The semantics for the ordering of units in the first list are defined by the individual queries.

  Every unit in this list is unique. No duplicates are returned; no two units A and B in the list have Is_Equal (A, B) = True.

- Inconsistent units list:

  The second list is made up of unit pairs.

  Each pairing defines an inconsistent semantic dependence relationship. The right unit of each pair semantically depends on the immediately preceding left unit. All rightmost units of each pair are always inconsistent, and will not appear in the consistent units list. The leftmost unit can be either consistent or inconsistent. If a unit that is the leftmost of a pair is consistent, then it also appears in the consistent units list; otherwise the unit is part of an inconsistent transitive relationship.

  The unit pairs are ordered such that there are no forward semantic dependencies between the inconsistent units. Each inconsistent unit's supporters always precede it in the list.

As an example, given four units, A withs B, B withs C, and C withs D; if D is replaced, the inconsistent list contains six units with the three pairs:

```
DC   CB   BA
```

The list indicates that units C, B, and A are inconsistent (the rightmost units of each pair). Semantic dependencies such as B depends on C also are indicated. The units C, B, and A are in an order that could be submitted to the compiler (a possible recompilation order).

If a unit is inconsistent because the source for the unit has been edited (or otherwise been made inconsistent by some action of the user or implementation) then the unit references Nil_Compilation_Unit as the cause of the inconsistency (e.g., (Nil A Nil B) is a list of two inconsistent units, neither of which can point to a third unit as the cause for their being inconsistent).

Implementation Permissions

An implementation is allowed to use Nil_Compilation_Unit value for the first unit of each pair if it cannot determine the supporting unit causing the inconsistent semantic dependence.

- Missing dependence list:

  The third list is made up of unit pairs. Each pairing consists of a unit followed by a missing related unit needed by the first unit. A missing unit is a required Compilation_Unit, with a known name, with a Unit_Kind that is either A_Nonexistent_Declaration or A_Nonexistent_Body.

  For example:

  Given a list containing the units: AB AC

  If Unit_Kind(B) = A_Nonexistent_Declaration and
  Unit_Kind(C) = A_Nonexistent_Body then

  It can be deduced that:
  A is missing a needed supporter B (A depends semantically on B).
  A is missing a needed related unit body C (depending on
  the kind for A, C can be A's required body or some subunit of A).

  A unit is reported as missing only if the Post-Compilation Rules of Ada determine it to be needed. Ada Standard 10.2.

- Circular dependence list:

  Circular dependencies between compilation units are provided in the fourth list. There may be more than one set of circular dependencies. The ordering of distinct sets in the list is implementation-defined. This list will never contain nonexistent units.

  The list is made up of unit pairs. The second unit of each pair depends semantically on the first unit. A circularity is established when the first unit of a pair also appears as the second unit of a later pair. (See the unit A in the example below; it is the first unit of the first pair and is the second unit of the third pair). The next set of circular dependent units, if any, starts with the next unit in the list (the unit D in the example below).

  For example:

  Given a list containing the units: AC CB BA DG GF FE ED

  It can be determined that there are two sets of circularly
  dependent units:
  {A, B, C} and {D, E, F, G}

  The dependencies are: A depends on B, B depends on C, C depends on A.

D depends on E, E depends on F, F depends on G, G depends on D.

Each circle of dependence is reported exactly once. It is not reported once for each unit in the circle.

12.2 constant Nil_Relationship

```
Nil_Relationship : constant Relationship :=
        (Consistent_Length   => 0,
         Inconsistent_Length => 0,
         Missing_Length      => 0,
         Circular_Length     => 0,
         Consistent          => Asis.Nil_Compilation_Unit_List,
         Inconsistent        => Asis.Nil_Compilation_Unit_List,
         Missing             => Asis.Nil_Compilation_Unit_List,
         Circular            => Asis.Nil_Compilation_Unit_List);
```

A Nil_Relationship is returned by all Compilation_Units.Relations functions when there is no relationship between the specified units.

12.3 function Semantic_Dependence_Order

To properly determine unit consistency, use one of the two semantic dependence queries: Elaboration_Order or Semantic_Dependence_Order. These queries return a value of the type Relationship, which contains lists of consistent, inconsistent, missing and circular units.

```
function Semantic_Dependence_Order
              (Compilation_Units : in Asis.Compilation_Unit_List;
               Dependent_Units   : in Asis.Compilation_Unit_List;
               The_Context       : in Asis.Context;
               Relation          : in Asis.Relation_Kinds)
             return Relationship;
```

Compilation_Units specifies a list of pertinent units. Dependent_Units specifies dependents used to limit the query. The_Context specifies a program Context for context. Relation specifies the relationship to query.

Produces a Relationship value containing compilation_unit elements related to the given Compilation_Units by the specified relation.

The compilation_unit elements in the consistent units list are ordered such that there are no forward semantic dependencies.

Dependent_Units are ignored unless the Relation is Descendants or Dependents. The union of units in the needed units of the Dependent_Units list provide a limiting context for the query. Only members of these needed units are present in the result.

If the Dependent_Units list is Is_Nil, the context for the search is the entire Context. The result of such a query is the full (unlimited) list of Dependents for the Compilation_Units.

All units in the result will have an Enclosing_Context value that Is_Identical to The_Context.

Compilation_Units and Dependent_Units expect a list of elements that each have one of the following Unit_Kinds:

A_Procedure
A_Function
A_Package
A_Generic_Procedure
A_Generic_Function
A_Generic_Package
A_Procedure_Instance

A_Function_Instance
A_Package_Instance
A_Procedure_Renaming
A_Function_Renaming
A_Package_Renaming
A_Generic_Procedure_Renaming
A_Generic_Function_Renaming
A_Generic_Package_Renaming
A_Procedure_Body
A_Function_Body
A_Package_Body
A_Procedure_Body_Subunit
A_Function_Body_Subunit
A_Package_Body_Subunit
A_Task_Body_Subunit
A_Protected_Body_Subunit
An_Unknown_Unit -- *See Implementation Permissions*

Raises ASIS_Inappropriate_Compilation_Unit with a Status of Value_Error for any unit that does not have one of these expected kinds.

The Semantic_Dependence_Order query should never raise an exception when processing inconsistent unit (sub)sets. This query is the only means for an application to know if a given unit is consistent with (some of) its supporters (dependents), and therefore the related semantic processing can give valuable results for this unit.

Implementation Permissions

The handling of An_Unknown_Unit is implementation specific. It can be possible to obtain Semantic Dependence Relationships when starting with a list containing one or more units that are An_Unknown_Unit. However, results may vary across ASIS implementations.

12.4 function Elaboration_Order

```
function Elaboration_Order
            (Compilation_Units : in Asis.Compilation_Unit_List;
             The_Context       : in Asis.Context)
           return Relationship;
```

Compilation_Units specifies a list of units to elaborate. The_Context specifies a program Context for context.

Produces, in elaboration order, a Relationship value containing compilation units required to elaborate the given compilation units.

The return value contains the set of ordered lists described above for the queries on Semantic Dependence Relationships. If the inconsistent, missing, and circular lists are empty, the consistent list will contain all units required to elaborate the arguments.

Implementation Permissions

The Relationship value may include any number of implementation-specific runtime support packages.

The first unit in the Consistent units list will always be the specification for package Standard. The list will contain all units required to elaborate the arguments.

Use the Context_Clause_Elements query to get pragma Elaborate elements for a compilation unit.

Compilation_Units expects a list of elements that each have one of the following Unit_Kinds:

A_Procedure
A_Function
A_Package
A_Generic_Procedure
A_Generic_Function
A_Generic_Package
A_Procedure_Instance
A_Function_Instance
A_Package_Instance
A_Procedure_Renaming
A_Function_Renaming
A_Package_Renaming
A_Generic_Procedure_Renaming
A_Generic_Function_Renaming
A_Generic_Package_Renaming
A_Procedure_Body
A_Function_Body
A_Package_Body
A_Procedure_Body_Subunit
A_Function_Body_Subunit
A_Package_Body_Subunit
A_Task_Body_Subunit
A_Protected_Body_Subunit
An_Unknown_Unit -- *See Implementation Permissions*

Raises ASIS_Inappropriate_Compilation_Unit with a Status of Value_Error for any unit that does not have one of these expected kinds.

Implementation Permissions

The handling of An_Unknown_Unit is implementation specific. It can be possible to obtain Semantic Dependence Relationships when starting with a list containing one or more units that are An_Unknown_Unit. However, results may vary across ASIS implementations.

Section 13: package Asis.Elements

The library package Asis.Elements shall exist. The package shall provide interfaces equivalent to those described in the following subclauses.

Asis.Elements encapsulates a set of queries that operate on all elements and some queries specific to A_Pragma elements.

13.1 function Unit_Declaration

```
function Unit_Declaration (Compilation_Unit : in Asis.Compilation_Unit)
                          return Asis.Declaration;
```

Compilation_Unit specifies the unit to query.

Returns the element representing the declaration of the compilation_unit.

Returns a Nil_Element if the unit is A_Nonexistent_Declaration, A_Nonexistent_Body, A_Configuration_Compilation, or An_Unknown_Unit.

Compilation_Unit expects a unit of any Unit_Kinds except Not_A_Unit.

Raises ASIS_Inappropriate_Compilation_Unit with a Status of Value_Error if Compilation_Unit has Unit_Kinds Not_A_Unit.

Returns an element that has one of the following Declaration_Kinds:

Not_A_Declaration
A_Function_Body_Declaration
A_Function_Declaration
A_Function_Instantiation
A_Generic_Function_Declaration
A_Generic_Package_Declaration
A_Generic_Procedure_Declaration
A_Package_Body_Declaration
A_Package_Declaration
A_Package_Instantiation
A_Procedure_Body_Declaration
A_Procedure_Declaration
A_Procedure_Instantiation
A_Task_Body_Declaration
A_Package_Renaming_Declaration
A_Procedure_Renaming_Declaration
A_Function_Renaming_Declaration
A_Generic_Package_Renaming_Declaration
A_Generic_Procedure_Renaming_Declaration
A_Generic_Function_Renaming_Declaration
A_Protected_Body_Declaration

13.2 function Enclosing_Compilation_Unit

```
function Enclosing_Compilation_Unit (Element : in Asis.Element)
                                    return Asis.Compilation_Unit;
```

Element specifies an Element whose Compilation_Unit is desired.

Returns the Compilation_Unit that contains the given Element.

Raises ASIS_Inappropriate_Element if the Element is a Nil_Element.

13.3 function Context_Clause_Elements

```
function Context_Clause_Elements
            (Compilation_Unit : in Asis.Compilation_Unit;
             Include_Pragmas  : in Boolean := False)
           return Asis.Context_Clause_List;
```

Compilation_Unit specifies the unit to query. Include_Pragmas specifies whether pragmas are to be returned.

Returns a list of with clauses, use clauses, and pragmas that explicitly appear in the context clause of the compilation unit, in their order of appearance.

Returns a Nil_Element_List if the Unit_Kind of the unit has A_Nonexistent_Declaration, A_Nonexistent_Body, or An_Unknown_Unit.

Implementation Requirements

All pragma Elaborate elements for this unit will appear in this list. Other pragmas will appear in this list, or in the Compilation_Pragmas list, or both.

Implementation Permissions

Implementors are encouraged to use this list to return all pragmas whose full effect is determined by their exact textual position. Pragmas that do not have placement dependencies may be returned in either list. Only pragmas that appear in the unit's context clause are returned by this query. All other pragmas, affecting the compilation of this unit, are available from the Compilation_Pragmas query.

Results of this query may vary across ASIS implementations. Some implementations normalize all multi-name with clauses and use clauses into an equivalent sequence of single-name with clauses and use clauses. Similarly, an implementation may retain only a single reference to a name that appeared more than once in the original context clause. Some implementors will return only pragma Elaborate elements in this list and return all other pragmas via the Compilation_Pragmas query.

Compilation_Unit expects a unit of any Unit_Kinds except Not_A_Unit.

Raises ASIS_Inappropriate_Compilation_Unit with a Status of Value_Error if Compilation_Unit has Unit_Kinds Not_A_Unit.

Returns a list of elements that each have one of the following Element_Kinds:

A_Pragma
A_Clause that has one of the following Clause_Kinds:
A_With_Clause
A_Use_Package_Clause

13.4 function Configuration_Pragmas

```
function Configuration_Pragmas (The_Context : in Asis.Context)
                               return Asis.Pragma_Element_List;
```

The_Context specifies the Context to query.

Returns a list of pragmas that apply to all future compilation_unit elements compiled into The_Context. Pragmas returned by this query should have appeared in a compilation that had no compilation_unit elements. To the extent that order is meaningful, the pragmas should be in their order of appearance in the compilation. (The order is implementation dependent, many pragmas have the same effect regardless of order.)

Returns a Nil_Element_List if there are no such configuration pragmas.

Returns a list of elements that each have the following Element_Kinds:

A_Pragma

13.5 function Compilation_Pragmas

```
function Compilation_Pragmas (Compilation_Unit : in Asis.Compilation_Unit)
                              return Asis.Pragma_Element_List;
```

Compilation_Unit specifies the unit to query.

Returns a list of pragmas that apply to the compilation of the unit. To the extent that order is meaningful, the pragmas should be in their order of appearance in the compilation. (The order is implementation dependent, many pragmas have the same effect regardless of order.)

There are two sources for the pragmas that appear in this list:

- Program unit pragmas appearing at the place of a compilation_unit. See Ada Standard 10.1.5(4).
- Configuration pragmas appearing before the first compilation_unit of a compilation. See Ada Standard 10.1.5(8).

This query does not return Elaborate pragmas from the unit context clause of the compilation unit; they do not apply to the compilation, only to the unit.

Use the Context_Clause_Elements query to obtain a list of all pragmas (including Elaborate pragmas) from the context clause of a compilation unit.

Pragmas from this query may be duplicates of some or all of the non-Elaborate pragmas available from the Context_Clause_Elements query. Such duplication is simply the result of the textual position of the pragma--globally effective pragmas may appear textually within the context clause of a particular unit, and be returned as part of the Context_Clause for that unit.

Ada predefined packages, such as package Standard, may or may not have pragmas available for processing by applications. The physical existence of a package Standard is implementation specific. The same is true for other Ada predefined packages, such as Ada.Text_Io and Ada.Direct_Io. The Origin query can be used to determine whether or not a particular unit is an Ada Predefined unit.

Returns a Nil_Element_List if the compilation unit:

- has no such applicable pragmas; or
- is an An_Unknown_Unit, A_Nonexistent_Declaration, or A_Nonexistent_Body.

Compilation_Unit expects a unit of any Unit_Kinds except Not_A_Unit.

Raises ASIS_Inappropriate_Compilation_Unit with a Status of Value_Error if Compilation_Unit has Unit_Kinds Not_A_Unit.

Returns a list of elements that each have the following Element_Kinds:

A_Pragma

13.6 function Element_Kind

```
function Element_Kind (Element : in Asis.Element)
                       return Asis.Element_Kinds;
```

Element specifies the element to query.

The hierarchy of Element_Kinds is shown in 3.7.1.

Returns the Element_Kinds value of Element. Returns Not_An_Element for a Nil_Element.

Element expects any kind of element.

13.7 function Pragma_Kind

```
function Pragma_Kind (Pragma_Element : in Asis.Pragma_Element)
                     return Asis.Pragma_Kinds;
```

Pragma_Element specifies the element to query.

Returns the Pragma_Kinds value of Pragma_Element. Returns Not_A_Pragma for any unexpected element such as a Nil_Element, A_Statement, or A_Declaration.

Pragma_Element expects an element that has the following Element_Kinds:

A_Pragma

13.8 function Defining_Name_Kind

```
function Defining_Name_Kind (Defining_Name : in Asis.Defining_Name)
                            return Asis.Defining_Name_Kinds;
```

Defining_Name specifies the element to query.

Returns the Defining_Name_Kinds value of the Defining_Name.

Returns Not_A_Defining_Name for any unexpected element such as a Nil_Element, A_Clause, or A_Statement.

Defining_Name expects an element that has the following Element_Kinds:

A_Defining_Name

13.9 function Declaration_Kind

```
function Declaration_Kind (Declaration : in Asis.Declaration)
                           return Asis.Declaration_Kinds;
```

Declaration specifies the element to query.

Returns the Declaration_Kinds value of the Declaration.

Returns Not_A_Declaration for any unexpected element such as a Nil_Element, A_Definition, or A_Statement.

Declaration expects an element that has the following Element_Kinds:

A_Declaration

13.10 function Has_Abstract

```
function Has_Abstract ( Element : in Asis.Element ) return Boolean;
```

Element specifies the element to query.

Returns True if the reserved word **abstract** appears in Element, and False otherwise.

Returns False for any unexpected element, including a Nil_Element.

Element expects an element that has one of the following Declaration_Kinds:

A_Formal_Procedure_Declaration
A_Formal_Function_Declaration
A_Function_Declaration

A_Private_Type_Declaration
A_Private_Extension_Declaration
A_Procedure_Declaration
A_Type_Declaration

or an element that has one of the following Definition_Kinds:

A_Private_Extension_Definition
A_Private_Type_Definition
A_Tagged_Private_Type_Definition
A_Type_Definition

or an element that has one of the following Formal_Type_Kinds:

A_Formal_Private_Type_Definition
A_Formal_Tagged_Private_Type_Definition
A_Formal_Derived_Type_Definition

13.11 function Has_Aliased

```
function Has_Aliased ( Element : in Asis.Element ) return Boolean;
```

Element specifies the Element to query.

Returns True if the reserved word **aliased** appears in Element, and False otherwise.

Returns False for any unexpected element, including a Nil_Element.

Element expects an element that has one of the following Declaration_Kinds:

A_Constant_Declaration
A_Deferred_Constant_Declaration
A_Return_Object_Specification
A_Variable_Declaration

or an element that has the following Definition_Kinds:

A_Component_Definition

13.12 function Has_Limited

```
function Has_Limited ( Element : in Asis.Element ) return Boolean;
```

Element specifies the Element to query.

Returns True if the reserved word **limited** appears in Element, and False otherwise.

Returns False for any unexpected element, including a Nil_Element.

Element expects an element that has the following Clause_Kinds: A_With_Clause

or an element that has one of the following Declaration_Kinds:

A_Type_Declaration
A_Private_Type_Declaration
A_Private_Extension_Declaration

or an element that has one of the following Definition_Kinds:

A_Type_Definition
A_Private_Type_Definition
A_Tagged_Private_Type_Definition
A_Private_Extension_Definition
An_Interface_Type_Definition

or an element that has one of the following Formal_Type_Kinds:

A_Formal_Private_Type_Definition
A_Formal_Tagged_Private_Type_Definition
A_Formal_Derived_Type_Definition

13.13 function Has_Private

```
function Has_Private ( Element : in Asis.Element ) return Boolean;
```

Element specifies the element to query.

Returns True if the reserved word **private** appears in Element, and False otherwise.

Returns False for any unexpected element, including a Nil_Element.

Element expects an element that has one of the following Declaration_Kinds:

A_Type_Declaration
A_Private_Type_Declaration

or an element that has one of the following Definition_Kinds:

A_Private_Extension_Definition
A_Private_Type_Definition
A_Tagged_Private_Type_Definition
A_Type_Definition

or an element that has one of the following Formal_Type_Kinds:

A_Formal_Private_Type_Definition
A_Formal_Tagged_Private_Type_Definition

or an element that has the following Clause_Kinds:

A_With_Clause

13.14 function Has_Protected

```
function Has_Protected ( Element : in Asis.Element ) return Boolean;
```

Element specifies the Element to query.

Returns True if the reserved word **protected** appears in Element, and False otherwise.

Returns False for any unexpected element, including a Nil_Element.

Element expects an element that has one of the following Definition_Kinds:

An_Interface_Type_Definition
A_Protected_Definition

or an element that has one of the following Declaration_Kinds:

A_Protected_Body_Declaration
A_Protected_Type_Declaration
A_Single_Protected_Declaration

13.15 function Has_Reverse

```
function Has_Reverse ( Element : in Asis.Element ) return Boolean;
```

Element specifies the Element to query.

Returns True if the reserved word **reverse** appears in Element, and False otherwise.

Returns False for any unexpected element, including a Nil_Element.

Element expects an element that has the following Declaration_Kinds:

A_Loop_Parameter_Specification

13.16 function Has_Synchronized

```
function Has_Synchronized ( Element : in Asis.Element ) return Boolean;
```

Element specifies the Element to query.

Returns True if the reserved word **synchronized** appears in Element, and False otherwise.

Returns False for any unexpected element, including a Nil_Element.

Element expects an element that has one of the following Definition_Kinds:

An_Interface_Type_Definition
A_Private_Extension_Definition

13.17 function Has_Tagged

```
function Has_Tagged ( Element : in Asis.Element ) return Boolean;
```

Element specifies the Element to query.

Returns True if the reserved word **tagged** appears in Element, and False otherwise.

Returns False for any unexpected element, including a Nil_Element.

Element expects an element that has one of the following Definition_Kinds:

A_Tagged_Incomplete_Type_Definition
A_Tagged_Private_Type_Definition
A_Tagged_Record_Type_Definition

or an element that has the following Type_Kinds:

A_Tagged_Record_Type_Definition

or an element that has the following Formal_Type_Kinds:

A_Formal_Tagged_Private_Type_Definition

13.18 function Has_Task

```
function Has_Task ( Element : in Asis.Element ) return Boolean;
```

Element specifies the Element to query.

Returns True if the reserved word **task** appears in Element, and False otherwise.

Returns False for any unexpected element, including a Nil_Element.

Element expects an element that has one of the following Definition_Kinds:

An_Interface_Type_Definition
A_Task_Definition

or an element that has one of the following Declaration_Kinds:

A_Task_Type_Declaration
A_Single_task_Declaration
A_Task_Body_Declaration

13.19 function Declaration_Origin

```
function Declaration_Origin (Declaration : in Asis.Declaration)
                             return Asis.Declaration_Origins;
```

Declaration specifies the Declaration to query.

Returns the Declaration_Origins value of the Declaration.

Returns Not_A_Declaration_Origin for any unexpected element such as a Nil_Element, A_Definition, or A_Clause.

Declaration expects an element that has the following Element_Kinds:

A_Declaration

13.20 function Mode_Kind

```
function Mode_Kind (Declaration : in Asis.Declaration)
                                return Asis.Mode_Kinds;
```

Declaration specifies the element to query.

Returns the Mode_Kinds value of the Declaration.

Returns A_Default_In_Mode for an access parameter.

Returns Not_A_Mode for any unexpected element such as a Nil_Element, A_Definition, or A_Statement.

Declaration expects an element that has one of the following Declaration_Kinds:

A_Parameter_Specification
A_Formal_Object_Declaration

13.21 function Default_Kind

```
function Default_Kind (Declaration : in Asis.Generic_Formal_Parameter)
                       return Asis.Subprogram_Default_Kinds;
```

Declaration specifies the element to query.

Returns the Subprogram_Default_Kinds value of the Declaration.

Returns Not_A_Declaration for any unexpected element such as a Nil_Element, A_Definition, or A_Statement.

Declaration expects an element that has one of the following Declaration_Kinds:

A_Formal_Function_Declaration
A_Formal_Procedure_Declaration

13.22 function Definition_Kind

```
function Definition_Kind (Definition : in Asis.Definition)
                          return Asis.Definition_Kinds;
```

Definition specifies the Definition to query.

Returns the Definition_Kinds value of the Definition.

Returns Not_A_Definition for any unexpected element such as a Nil_Element, A_Statement, or A_Declaration.

Definition expects an element that has the following Element_Kinds:

A_Definition

13.23 function Type_Kind

```
function Type_Kind (Definition : in Asis.Type_Definition)
                                 return Asis.Type_Kinds;
```

Definition specifies the Type_Definition to query.

Returns the Type_Kinds value of the Definition.

Returns Not_A_Type_Definition for any unexpected element such as a Nil_Element, A_Statement, or A_Declaration.

Definition expects an element that has the following Definition_Kinds:

A_Type_Definition

13.24 function Formal_Type_Kind

```
function Formal_Type_Kind
               (Definition : in Asis.Formal_Type_Definition)
                return Asis.Formal_Type_Kinds;
```

Definition specifies the Formal_Type_Definition to query.

Returns the Formal_Type_Kinds value of the Definition.

Returns Not_A_Formal_Type_Definition for any unexpected element such as a Nil_Element, A_Statement, or A_Declaration.

Definition expects an element that has the following Definition_Kinds:

A_Formal_Type_Definition

13.25 function Access_Type_Kind

```
function Access_Type_Kind
               (Definition : in Asis.Access_Type_Definition)
                return Asis.Access_Type_Kinds;
```

Definition specifies the Access_Type_Definition to query.

Returns the Access_Type_Kinds value of the Definition.

Returns Not_An_Access_Type_Definition for any unexpected element such as a Nil_Element, A_Statement, or A_Declaration.

Definition expects an element that has the following Type_Kinds:

An_Access_Type_Definition

13.26 function Has_Null_Exclusion

```
function Has_Null_Exclusion -- 3.2.2(3/2), 3.7(5/2), 3.10 (2/2,6/2),
                            -- 6.1(13/2,15/2), 8.5.1(2/2), 12.4(2/2)
     (Element : in Asis.Element)
      return Boolean;
```

Element specifies the element to query.

Returns True if the argument element has a null_exclusion specifier, and returns False otherwise (including for any unexpected element).

Element expects an element that has one of the following Definition_Kinds:

A_Type_Definition
An_Access_Definition
A_Subtype_Indication

or Element expects an that has the following Type_Kinds:

An_Access_Type_Definition

or Element expects an that has one of the following Declaration_Kinds:

A_Discriminant_Specification
A_Parameter_Specification
A_Formal_Object_Declaration
An_Object_Renaming_Declaration

13.27 function Access_Definition_Kind

```
function Access_Definition_Kind -- 3.3.1(2) / 3.10(6)
     (Definition : in Asis.Definition)
      return Asis.Access_Definition_Kinds;
```

Definition specifies the Definition to query.

Returns the Access_Definition_Kinds value of the Definition.

Returns Not_An_Access_Definition for any unexpected element such as a Nil_Element, A_Statement, or A_Declaration.

Definition expects an element that has the following Element_Kinds:

An_Access_Definition

13.28 function Interface_Kind

```
function Interface_Kind
     (Definition : in Asis.Definition)
      return Asis.Interface_Kinds;
```

Definition specifies the Definition to query.

Returns the Interface_Kinds value of the Definition.

Returns Not_An_Interface for any unexpected element such as a Nil_Element, A_Statement, or A_Declaration.

Definition expects an element that has the following Definition_Kinds:

A_Type_Definition that has one of the following Type_Kinds:
  An_Interface_Type_Definition
A_Formal_Type_Definition that has the following Formal_Type_Kinds:
  A_Formal_Interface_Type_Definition

13.29 function Root_Type_Kind

```
function Root_Type_Kind
               (Definition : in Asis.Root_Type_Definition)
                return Asis.Root_Type_Kinds;
```

Definition specifies the Root_Type_Definition to query.

Returns the Root_Type_Kinds value of the Definition.

Returns Not_A_Root_Type_Definition for any unexpected element such as a Nil_Element, A_Statement, or A_Declaration.

Definition expects an element that has the following Type_Kinds:

A_Root_Type_Definition

13.30 function Constraint_Kind

```
function Constraint_Kind
               (Definition : in Asis.Constraint)
                return Asis.Constraint_Kinds;
```

Definition specifies the constraint to query.

Returns the Constraint_Kinds value of the Definition.

Returns Not_A_Constraint for any unexpected element such as a Nil_Element, A_Statement, or A_Declaration.

Definition expects an element that has the following Definition_Kinds:

A_Constraint

13.31 function Discrete_Range_Kind

```
function Discrete_Range_Kind
               (Definition : in Asis.Discrete_Range)
                return Asis.Discrete_Range_Kinds;
```

Definition specifies the discrete_range to query.

Returns the Discrete_Range_Kinds value of the Definition.

Returns Not_A_Discrete_Range for any unexpected element such as a Nil_Element, A_Statement, or A_Declaration.

Definition expects an element that has one of the following Definition_Kinds:

A_Discrete_Subtype_Definition
A_Discrete_Range

13.32 function Expression_Kind

```
function Expression_Kind (Expression : in Asis.Expression)
                          return Asis.Expression_Kinds;
```

Expression specifies the Expression to query.

Returns the Expression_Kinds value of the Expression.

Returns Not_An_Expression for any unexpected element such as a Nil_Element, A_Statement, or A_Declaration.

Expression expects an element that has the following Element_Kinds:

An_Expression

13.33 function Operator_Kind

```
function Operator_Kind (Element : in Asis.Element)
                        return Asis.Operator_Kinds;
```

Element specifies the Element to query.

Returns the Operator_Kinds value of the element.

Returns Not_An_Operator for any unexpected element such as a Nil_Element, A_Statement, or A_Declaration.

Element expects an element that has the following Defining_Name_Kinds:

A_Defining_Operator_Symbol

or that has the following Expression_Kinds:

An_Operator_Symbol

13.34 function Attribute_Kind

```
function Attribute_Kind (Expression : in Asis.Expression)
                         return Asis.Attribute_Kinds;
```

Expression specifies the Expression to query.

Returns the Attribute_Kinds value of the Expression.

Returns Not_An_Attribute for any unexpected element such as a Nil_Element, A_Statement, or A_Declaration.

Expression expects an element that has the following Expression_Kinds:

An_Attribute_Reference

13.35 function Association_Kind

```
function Association_Kind (Association : in Asis.Association)
                           return Asis.Association_Kinds;
```

Association specifies the Association to query.

Returns the Association_Kinds value of the Association.

Returns Not_An_Association for any unexpected element such as a Nil_Element, A_Statement, or A_Declaration.

Association expects an element that has the following Element_Kinds:

An_Association

13.36 function Statement_Kind

```
function Statement_Kind (Statement : in Asis.Statement)
                         return Asis.Statement_Kinds;
```

Statement specifies the element to query.

Returns the Statement_Kinds value of the statement.

Returns Not_A_Statement for any unexpected element such as a Nil_Element, A_Definition, or A_Declaration.

Statement expects an element that has the following Element_Kinds:

A_Statement

13.37 function Path_Kind

```
function Path_Kind (Path : in Asis.Path) return Asis.Path_Kinds;
```

Path specifies the Path to query.

Returns the Path_Kinds value of the Path.

Returns Not_A_Path for any unexpected element such as a Nil_Element, A_Statement, or A_Declaration.

Path expects an element that has the following Element_Kinds:

A_Path

13.38 function Clause_Kind

```
function Clause_Kind (Clause : in Asis.Clause) return Asis.Clause_Kinds;
```

Clause specifies the element to query.

Returns the Clause_Kinds value of the Clause.

Returns Not_A_Clause for any unexpected element such as a Nil_Element, A_Definition, or A_Declaration.

Clause expects an element that has the following Element_Kinds:

A_Clause

13.39 function Aspect_Clause_Kind

```
function Aspect_Clause_Kind
               (Clause : in Asis.Aspect_Clause)
                return Asis.Aspect_Clause_Kinds;
```

Clause specifies the element to query.

Returns the Aspect_Clause_Kinds value of the Clause.

Returns Not_An_Aspect_Clause for any unexpected element such as a Nil_Element, A_Definition, or A_Declaration.

Clause expects an element that has the following Clause_Kinds:

An_Aspect_Clause

13.40 function Is_Nil (element)

```
function Is_Nil (Right : in Asis.Element) return Boolean;
```

Right specifies the element to check.

Returns True if the program element is the Nil_Element and returns False otherwise.

13.41 function Is_Nil (element list)

```
function Is_Nil (Right : in Asis.Element_List) return Boolean;
```

Right specifies the element list to check.

Returns True if the element list has a length of zero and returns False otherwise.

13.42 function Is_Equal (element)

```
function Is_Equal (Left  : in Asis.Element;
                   Right : in Asis.Element) return Boolean;
```

Left specifies the left element to compare. Right specifies the right element to compare.

Returns True if Left and Right represent the same physical element from the same physical compilation unit, and returns False otherwise. The two elements may or may not be from the same open ASIS Context variable.

A True result implies:

```
Is_Equal (Enclosing_Compilation_Unit (Left),
          Enclosing_Compilation_Unit (Right)) = True
```

13.43 function Is_Identical (element)

```
function Is_Identical (Left  : in Asis.Element;
                       Right : in Asis.Element) return Boolean;
```

Left specifies the left element. Right specifies the right element.

Returns True if Left and Right represent the same physical element from the same physical compilation unit from the same open ASIS Context variable, and returns False otherwise.

A True result implies:

```
Is_Identical (Enclosing_Compilation_Unit (Left),
              Enclosing_Compilation_Unit (Right)) = True
```

13.44 function Is_Part_Of_Implicit

```
function Is_Part_Of_Implicit (Element : in Asis.Element) return Boolean;
```

Element specifies the element to query.

Returns True for any Element that is, or that forms part of, any implicitly declared or specified program Element structure.

Returns True for any implicit generic child unit specifications or their subcomponents. Ada Standard 10.1.1(19).

Returns False for a Nil_Element, or any Element that correspond to text which was specified explicitly (typed, entered, written).

Generic instance specifications and bodies, while implicit, are treated as a special case. These elements will not normally test as Is_Part_Of_Implicit. Rather, they are Is_Part_Of_Instance. They only test as Is_Part_Of_Implicit if one of the following rules applies. This is done so that it is possible to determine whether a declaration, which happens to occur within an instance, is an implicit result of another declaration which occurs explicitly within the generic template.

Implicit Elements are those that represent these portions of the Ada language:

- Ada Standard 4.5.(9)
  - All predefined operator declarations and their component elements are Is_Part_Of_Implicit
- Ada Standard 3.4(16)
  - Implicit predefined operators of the derived type.
- Ada Standard 3.4(17-22)
  - Implicit inherited subprogram declarations and their component elements are Is_Part_Of_Implicit
- Ada Standard 6.4(9) and 12.3(7)
  - Implicit actual parameter expressions (defaults).
  - The A_Parameter_Association that includes a defaulted parameter value Is_Normalized and also Is_Part_Of_Implicit. The Formal_Parameter and the Actual_Parameter values from such Associations are not Is_Part_Of_Implicit unless they are from default initializations for an inherited subprogram declaration and have an Enclosing_Element that is the parameter

specification of the subprogram declaration. (Those elements are shared with (were created by) the original subprogram declaration, or they are naming expressions representing the actual generic subprogram selected at the place of an instantiation for A_Box_Default.)

- All A_Parameter_Association Kinds from a Normalized list are Is_Part_Of_Implicit.

- Ada Standard 6.6 (6)
  - Inequality operator declarations for limited private types are Is_Part_Of_Implicit.
  - Depending on the ASIS implementation, a "/=" appearing in the compilation may result in a **not** and an "=" in the internal representation. These two elements test as Is_Part_Of_Implicit because they do not represent text from the original compilation text.
- Ada Standard 12.3 (16)
  - implicit generic instance specifications and bodies are not Is_Part_Of_Implicit; they are Is_Part_Of_Instance and are only implicit if some other rule makes them so.

13.45 function Is_Part_Of_Inherited

```
function Is_Part_Of_Inherited (Element : in Asis.Element) return Boolean;
```

Element specifies the element to query.

Returns True for any Element that is, or that forms part of, an inherited primitive subprogram declaration.

Returns False for any other Element including a Nil_Element.

13.46 function Is_Part_Of_Instance

```
function Is_Part_Of_Instance (Element : in Asis.Element) return Boolean;
```

Element specifies the element to test.

Returns True if the Element is part of an implicit generic specification instance or an implicit generic body instance.

Returns False for explicit, inherited, and predefined Elements that are not the result of a generic expansion.

Returns False for any implicit generic child unit specifications or their subcomponents. Ada Standard 10.1.1(19).

Returns False for a Nil_Element.

Instantiations are not themselves Is_Part_Of_Instance unless they are encountered while traversing a generic instance.

13.47 function Is_Prefix_Notation

```
function Is_Prefix_Notation (Call : in Asis.Element) return Boolean;
```

Call specifies the call to query.

Returns True if the prefix of the call is a prefixed view of the subprogram (see Ada Standard 4.1.3(9.2/2)), and returns False otherwise.

Call expects an element that has the following Statement_Kinds:

A_Procedure_Call_Statement

or an element that has the following Expression_Kinds:

A_Function_Call

Raises ASIS_Inappropriate_Element with a Status of Value_Error for any element that does not have one of these expected kinds.

13.48 function Is_Null_Procedure

```
function Is_Null_Procedure
     (Element : in Asis.Element)
      return Boolean;
```

Element specifies the element to query.

Returns True for a declaration of a procedure or formal procedure that is declared as null.

Returns False for any other Element including a Nil_Element.

Element expects an element that has the following Element_Kinds:

A_Declaration

that has one of the following Declaration_Kinds:

A_Procedure_Declaration
A_Formal_Procedure_Declaration

13.49 function Is_Abstract_Subprogram

```
function Is_Abstract_Subprogram
     (Element : in Asis.Element)
      return Boolean;
```

Element specifies the element to query.

Returns True for a declaration of a subprogram or formal subprogram that is declared as abstract.

Returns False for any other Element including a Nil_Element.

Element expects an element that has the following Element_Kinds:

A_Declaration

that has one of the following Declaration_Kinds:

A_Procedure_Declaration
A_Function_Declaration
A_Formal_Procedure_Declaration
A_Formal_Function_Declaration

13.50 function Enclosing_Element

```
function Enclosing_Element (Element : in Asis.Element) return Asis.Element;

function Enclosing_Element (Element                    : in Asis.Element;
                            Expected_Enclosing_Element : in Asis.Element)
                         return Asis.Element;
```

Element specifies the element to query. Expected_Enclosing_Element specifies an enclosing element expected to contain the element.

Returns the Element that immediately encloses the given element. This query is intended to exactly reverse any single parent-to-child element traversal. For any structural query

that returns a subcomponent of an element (or that returns a list of subcomponent elements), the original element can be determined by passing the subcomponent element to this query.

Returns a Nil_Element if:

- the element is the declaration part of a compilation unit (Unit_Declaration).
- the element is with clause or use clause of a context clause (Context_Clause_Elements).
- the element is a pragma for a compilation unit (Compilation_Pragmas and Context_Clause_Elements).

Use Enclosing_Compilation_Unit to get the enclosing compilation unit for any element value other than Nil_Element.

Raises ASIS_Inappropriate_Element if the Element is a Nil_Element.

Examples

- Given a A_Declaration/A_Full_Type_Declaration in the declarative region of a block statement, returns the A_Statement/A_Block_Statement Element that encloses the type declaration.
- Given A_Statement, from the sequence of statements within a loop statement, returns the enclosing A_Statement/A_Loop_Statement.
- Given the An_Expression/An_Identifier selector from an expanded name, returns the An_Expression/A_Selected_Component that represents the combination of the prefix, the dot, and the selector.
- Given the A_Declaration corresponding to the implicit redeclaration of a child generic for an instantiated parent generic, returns the expanded generic specific template from the parent generic instantiation corresponding to any implicit generic child unit specification given as an argument. Ada Standard 10.1.1(19).

13.51 function Pragmas

```
function Pragmas (The_Element : in Asis.Element)
                  return Asis.Pragma_Element_List;
```

The_Element specifies the element to query.

Returns the list of pragmas, in their order of appearance, that appear directly within the given The_Element. Returns only those pragmas that are immediate component elements of the given The_Element. Pragmas embedded within other component elements are not returned. For example, returns the pragmas in a package specification, in the statement list of a loop, or in a record component list.

This query returns exactly those pragmas that are returned by the various queries, that accept these same argument kinds, and that return Declaration_List and Statement_List, where the inclusion of Pragmas is controlled by an Include_Pragmas parameter.

Returns a Nil_Element_List if there are no pragmas.

The_Element expects an element that has one of the following Element_Kinds:

A_Path — (pragmas from the statement list +
pragmas immediately preceding the reserved word **when** of the first alternative)
An_Exception_Handler — (pragmas from the statement list +
pragmas immediately preceding the reserved word **when** of the first exception han
dler)

or an element that has one of the following Declaration_Kinds:

A_Procedure_Body_Declaration — (pragmas from declarative region + statements)
A_Function_Body_Declaration — (pragmas from declarative region + statements)
A_Package_Declaration — (pragmas from visible + private declarative regions)
A_Package_Body_Declaration — (pragmas from declarative region + statements)
A_Task_Body_Declaration — (pragmas from declarative region + statements)
A_Protected_Body_Declaration — (pragmas from declarative region)
An_Entry_Body_Declaration — (pragmas from declarative region + statements)
A_Generic_Procedure_Declaration — (pragmas from formal declarative region)
A_Generic_Function_Declaration — (pragmas from formal declarative region)
A_Generic_Package_Declaration —
(pragmas from formal + visible + private declarative regions)

or an element that has one of the following Definition_Kinds:

A_Record_Definition — (pragmas from the component list)
A_Variant_Part —
(pragmas immediately preceding the first reserved word **when** + between variants)
A_Variant — (pragmas from the component list)
A_Task_Definition — (pragmas from visible + private declarative regions)
A_Protected_Definition — (pragmas from visible + private declarative regions)

or an element that has one of the following Statement_Kinds:

A_Loop_Statement — (pragmas from statement list)
A_While_Loop_Statement — (pragmas from statement list)
A_For_Loop_Statement — (pragmas from statement list)
A_Block_Statement — (pragmas from declarative region + statements)
An_Accept_Statement — (pragmas from statement list)

or an element that has the following Aspect_Clause_Kinds:

A_Record_Representation_Clause — (pragmas from component specifications)

Raises ASIS_Inappropriate_Element with a Status of Value_Error for any element that does not have one of these expected kinds.

Returns a list of elements that each have the following Element_Kinds:

A_Pragma

13.52 function Corresponding_Aspect_Pragmas

```
function Corresponding_Aspect_Pragmas (Defining_Name : in Asis.Defining_Name)
                                       return Asis.Pragma_Element_List;
```

Defining_Name specifies the defining name to query.

Returns all aspect pragma elements that apply to the named entity.

The pragmas returned may be the pragmas applying to a parent type if the name refers to type which is a derived type with no explicit aspect pragma. These pragmas are not Is_Part_Of_Implicit, they are the A_Pragma elements specified in conjunction with the declaration of the parent type.

Returns a Nil_Element_List if there are no aspect pragmas that apply to the named entity.

Defining_Name expects an element that has the following Element_Kinds:

A_Defining_Name

Returns a list of elements that each have the following Element_Kinds:

A_Pragma

13.53 function Pragma_Name_Image

```
function Pragma_Name_Image
         (Pragma_Element : in Asis.Pragma_Element) return Program_Text;
```

Pragma_Element specifies the element to query.

Returns the program text image of the simple name of the pragma.

The case of names returned by this query may vary between implementors. Implementors are encouraged, but not required, to return names in the same case as was used in the original compilation text.

Pragma_Element expects an element that has the following Element_Kinds:

A_Pragma

Raises ASIS_Inappropriate_Element with a Status of Value_Error for any element that does not have one of these expected kinds.

13.54 function Pragma_Argument_Associations

```
function Pragma_Argument_Associations
         (Pragma_Element : in Asis.Pragma_Element)
             return Asis.Association_List;
```

Pragma_Element specifies the element to query.

Returns a list of the Pragma_Argument_Associations of the pragma, in their order of appearance.

Pragma_Element expects an element that has the following Element_Kinds:

A_Pragma

Raises ASIS_Inappropriate_Element with a Status of Value_Error for any element that does not have one of these expected kinds.

Returns a list of elements that each have the following Element_Kinds:

A_Pragma_Argument_Association

13.55 function Debug_Image (element)

```
function Debug_Image (Element : in Asis.Element) return Wide_String;
```

Element specifies the program element to convert.

Returns a string value containing implementation-defined debug information associated with the element.

The return value uses Asis.Text.Delimiter_Image to separate the lines of multi-line results. The return value does not end with Asis.Text.Delimiter_Image.

These values are intended for two purposes. They are suitable for inclusion in problem reports sent to the ASIS implementor. They can be presumed to contain information useful when debugging the implementation itself. They are also suitable for use by the ASIS application when printing simple application debugging messages during application development. They are intended to be, to some worthwhile degree, intelligible to the user.

13.56 function Hash

```
function Hash (Element : in Asis.Element) return Asis.ASIS_Integer;
```

The purpose of the hash function is to provide a convenient name for an object of type Asis.Element in order to facilitate application defined I/O and/or other application defined processing.

The hash function maps Asis.Element objects into N discrete classes ("buckets") of objects. A good hash function is uniform across its range. It is important to note that the distribution of objects in the application's domain will affect the distribution of the hash function. A good hash measured against one domain will not necessarily be good when fed objects from a different set.

A continuous uniform hash can be divided by any N and provide a uniform distribution of objects to each of the N discrete classes. A hash value is not unique for each hashed Asis.Element. The application is responsible for handling name collisions of the hashed value.

The hash function returns a hashed value of type ASIS_Integer. If desired, a user could easily map ASIS_Integer'Range to any smaller range for the hash based on application constraints (i.e., the application implementor can tune the time-space tradeoffs by choosing a small table, implying slower lookups within each "bucket", or a large table, implying faster lookups within each "bucket").

Section 14: package Asis.Iterator

The library package Asis.Iterator shall exist. The package shall provide interfaces equivalent to those described in the following subclauses.

Asis.Iterator encapsulates the generic procedure Traverse_Element which allows an ASIS application to perform an iterative traversal of a logical syntax tree. It requires the use of two generic procedures, Pre_Operation, which identifies processing for the traversal, and Post_Operation, which identifies processing after the traversal. The State_Information allows processing state to be passed during the iteration of Traverse_Element.

Package Asis.Iterator is established as a child package to highlight the iteration capability and to facilitate the translation of ASIS to IDL.

14.1 procedure Traverse_Element

```
generic
   type State_Information is limited private;
   with procedure Pre_Operation
                    (Element : in     Asis.Element;
                     Control : in out Traverse_Control;
                     State   : in out State_Information) is <>;
   with procedure Post_Operation
                    (Element : in     Asis.Element;
                     Control : in out Traverse_Control;
                     State   : in out State_Information) is <>;
procedure Traverse_Element
              (Element : in     Asis.Element;
               Control : in out Traverse_Control;
               State   : in out State_Information);
```

Element specifies the initial element in the traversal. Control specifies what next to do with the traversal. State specifies other information for the traversal.

Traverses the element and all its component elements, if any. Component elements are all elements that can be obtained by a combination of the ASIS structural queries appropriate for the given element.

If an element has one or more component elements, each is called a child element. An element's parent element is its Enclosing_Element. Children with the same parent are sibling elements. The type Traverse_Control uses the terms children and siblings to control the traversal.

For each element, the formal procedure Pre_Operation is called when first visiting the element. Each of that element's children are then visited and finally the formal procedure Post_Operation is called for the element.

The order of Element traversal is in terms of the textual representation of the Elements. Elements are traversed in left-to-right and top-to-bottom order.

Traversal of Implicit Elements:

Implicit elements are not traversed by default. However, they may be explicitly queried and then passed to the traversal instance. Implicit elements include implicit predefined operator declarations, implicit inherited subprogram declarations, implicit expanded generic specifications and bodies, default expressions supplied to procedure, function, and entry calls, etc.

Applications that wish to traverse these implicit Elements shall query for them at the appropriate places in a traversal and then recursively call their instantiation of the

traversal generic. (Implicit elements provided by ASIS do not cover all possible Ada implicit constructs. For example, implicit initializations for variables of an access type are not provided by ASIS.)

Traversal of Association lists:

Argument and association lists for procedure calls, function calls, entry calls, generic instantiations, and aggregates are traversed in their unnormalized forms, as if the Normalized parameter was False for those queries. Implementations that always normalize certain associations may return Is_Normalized associations. See the Implementation Permissions for the queries Discriminant_Associations, Generic_Actual_Part, Call_Statement_Parameters, Record_Component_Associations, or Function_Call_Parameters.

Applications that wish to explicitly traverse normalized associations can do so by querying the appropriate locations in order to obtain the normalized list. The list can then be traversed by recursively calling the traverse instance. Once that sub-traversal is finished, the Control parameter can be set to Abandon_Children to skip processing of the unnormalized argument list.

Traversal can be controlled with the Control parameter.

A call to an instance of Traverse_Element will not result in calls to Pre_Operation or Post_Operation unless Control is set to Continue.

The subprograms matching Pre_Operation and Post_Operation can set their Control parameter to affect the traversal:

- Continue
  - Continues the normal depth-first traversal.
- Abandon_Children
  - Prevents traversal of the current element's children.
  - If set in a Pre_Operation, traversal picks up with the next sibling element of the current element.
  - If set in a Post_Operation, this is the same as Continue, all children will already have been traversed. Traversal picks up with the Post_Operation of the parent.
- Abandon_Siblings
  - Prevents traversal of the current element's children and remaining siblings.
  - If set in a Pre_Operation, this abandons the associated Post_Operation for the current element. Traversal picks up with the Post_Operation of the parent.
  - If set in a Post_Operation, traversal picks up with the Post_Operation of the parent.
- Terminate_Immediately
  - Does exactly that.

Raises ASIS_Inappropriate_Element with a Status of Value_Error if the element is a Nil_Element.

Section 15: package Asis.Declarations

The library package Asis.Declarations shall exist. The package shall provide interfaces equivalent to those described in the following subclauses.

Asis.Declarations encapsulates a set of queries that operate on A_Defining_Name and A_Declaration elements.

15.1 function Names

```
function Names (Declaration : in Asis.Declaration)
               return Asis.Defining_Name_List;
```

Declaration specifies the element to query.

Returns a list of names defined by the declaration, in their order of appearance. Declarations that define a single name will return a list of length one.

Returns Nil_Element_List for A_Declaration Elements representing the (implicit) declarations of universal and root numeric types (that is, if Type_Kind (Type_Declaration_View (Declaration)) = A_Root_Type_Definition).

Function designators that define operators are A_Defining_Operator_Symbol.

Declaration expects an element that has the following Element_Kinds:

A_Declaration

Raises ASIS_Inappropriate_Element with a Status of Value_Error for any element that does not have one of these expected kinds.

Returns a list of elements that have the following Element_Kinds:

A_Defining_Name

Examples

```
type Foo is (Pooh, Baah);
   -- Returns a list containing one A_Defining_Name: Foo.
One, Uno : constant Integer := 1;
   -- Returns a list of two A_Defining_Name elements: One and Uno.
```

15.2 function Defining_Name_Image

```
function Defining_Name_Image
        (Defining_Name : in Asis.Defining_Name) return Program_Text;
```

Defining_Name specifies the element to query.

Returns the program text image of the name. Embedded quotes (for operator designator strings) are doubled.

A_Defining_Identifier elements are simple identifier names "Abc" (name Abc).

A_Defining_Operator_Symbol elements have names with embedded quotes """abs""" (function "**abs**").

A_Defining_Character_Literal elements have names with embedded apostrophes "'x'" (literal 'x').

A_Defining_Enumeration_Literal elements have simple identifier names "Blue" (literal Blue). If A_Defining_Enumeration_Literal element is of type Character or Wide_Character but does not have a graphical presentation, then the result is implementation-dependent.

A_Defining_Expanded_Name elements are prefix.selector names "A.B.C" (name A.B.C).

The case of names returned by this query may vary between implementors. Implementors are encouraged, but not required, to return names in the same case as was used in the original compilation text.

The Defining_Name_Image of a label_statement_identifier does not include the enclosing "<<" and ">>" that form the label syntax. Similarly, the Defining_Name_Image of an identifier for a loop_statement or block_statement does not include the trailing colon that forms the loop name syntax. Use Asis.Text.Element_Image or Asis.Text.Lines queries to obtain these syntactic constructs and any comments associated with them.

Defining_Name expects an element that has the following Element_Kinds:

A_Defining_Name

Raises ASIS_Inappropriate_Element with a Status of Value_Error for any element that does not have one of these expected kinds.

15.3 function Position_Number_Image

```
function Position_Number_Image
        (Defining_Name : in Asis.Defining_Name) return Wide_String;
```

Expression specifies the literal expression to query.

Returns the program text image of the position number of the value of the enumeration literal.

The program text returned is the image of the universal_integer value that is returned by the attribute 'Pos if it were applied to the value. For example: Integer'Image(Color'Pos(Blue)).

Defining_Name expects an element that is one of the following Defining_Name_Kinds:

A_Defining_Character_Literal
A_Defining_Enumeration_Literal

Raises ASIS_Inappropriate_Element with a Status of Value_Error for any element that does not have one of these expected kinds.

15.4 function Representation_Value_Image

```
function Representation_Value_Image
        (Defining_Name : in Asis.Defining_Name) return Wide_String;
```

Expression specifies the literal expression to query.

Returns the string image of the internal code for the enumeration literal.

If an aspect_clause is defined for the enumeration type then the string returned is the Integer'Wide_Image of the corresponding value given in the enumeration_aggregate. Otherwise, the string returned is the same as the Position_Number_Image.

Defining_Name expects an element that is one of the following Defining_Name_Kinds:

A_Defining_Character_Literal
A_Defining_Enumeration_Literal

Raises ASIS_Inappropriate_Element with a Status of Value_Error for any element that does not have one of these expected kinds.

15.5 function Defining_Prefix

```
function Defining_Prefix (Defining_Name : in Asis.Defining_Name)
                  return Asis.Name;
```

Defining_Name specifies the element to query.

Returns the element that forms the prefix of the name. The prefix is the name to the left of the rightmost 'dot' in the expanded name. The Defining_Prefix of A.B is A, and of A.B.C is A.B.

Defining_Name expects an element that has the following Defining_Name_Kinds:

A_Defining_Expanded_Name

Raises ASIS_Inappropriate_Element with a Status of Value_Error for any element that does not have one of these expected kinds.

Returns an element that has one of the following Expression_Kinds:

An_Identifier
A_Selected_Component

15.6 function Defining_Selector

```
function Defining_Selector (Defining_Name : in Asis.Defining_Name)
                    return Asis.Defining_Name;
```

Defining_Name specifies the element to query.

Returns the element that forms the selector of the name. The selector is the name to the right of the rightmost 'dot' in the expanded name. The Defining_Selector of A.B is B, and of A.B.C is C.

Defining_Name expects an element that has the following Defining_Name_Kinds:

A_Defining_Expanded_Name

Raises ASIS_Inappropriate_Element with a Status of Value_Error for any element that does not have one of these expected kinds.

Returns an element that has the following Defining_Name_Kinds:

A_Defining_Identifier

15.7 function Discriminant_Part

```
function Discriminant_Part (Declaration : in Asis.Declaration)
                      return Asis.Definition;
```

Declaration specifies the type declaration to query.

Returns the discriminant_part, if any, from the type_declaration or formal_type_declaration.

Returns a Nil_Element if the Declaration has no explicit discriminant_part.

Declaration expects an element that has one of the following Declaration_Kinds:

An_Ordinary_Type_Declaration
A_Task_Type_Declaration
A_Protected_Type_Declaration
An_Incomplete_Type_Declaration
A_Private_Type_Declaration
A_Private_Extension_Declaration

A_Formal_Type_Declaration

Raises ASIS_Inappropriate_Element with a Status of Value_Error for any element that does not have one of these expected kinds.

Returns an element that has one of the following Definition_Kinds:

Not_A_Definition
An_Unknown_Discriminant_Part
A_Known_Discriminant_Part

15.8 function Type_Declaration_View

```
function Type_Declaration_View (Declaration : in Asis.Declaration)
              return Asis.Definition;
```

Declaration specifies the declaration element to query.

Returns the definition characteristics that form the view of the type_declaration. The view is the remainder of the declaration following the reserved word **is**.

For a full_type_declaration, returns the type_definition, task_definition, or protected_definition following the reserved word **is** in the declaration.

Returns a Nil_Element for a task_type_declaration that has no explicit task_definition.

For a private_type_declaration or private_extension_declaration, returns the definition element representing the private declaration view.

For an incomplete_type_declaration, returns the definition element representing the incomplete declaration view.

For a subtype_declaration, returns the subtype_indication.

For a formal_type_declaration, returns the formal_type_definition.

Declaration expects an element that has one of the following Declaration_Kinds:

An_Ordinary_Type_Declaration
A_Task_Type_Declaration
A_Protected_Type_Declaration
A_Private_Type_Declaration
A_Private_Extension_Declaration
A_Subtype_Declaration
A_Formal_Type_Declaration
An_Incomplete_Type_Declaration

Raises ASIS_Inappropriate_Element with a Status of Value_Error for any element that does not have one of these expected kinds.

Returns an element that has one of the following Definition_Kinds:

Not_A_Definition
A_Type_Definition
A_Subtype_Indication
An_Incomplete_Type_Definition
A_Tagged_Incomplete_Type_Definition
A_Private_Type_Definition
A_Tagged_Private_Type_Definition
A_Private_Extension_Definition
A_Task_Definition
A_Protected_Definition
A_Formal_Type_Definition

15.9 function Object_Declaration_Subtype

```
function Object_Declaration_Subtype (Declaration : in Asis.Declaration)
              return Asis.Definition;
```

Declaration specifies the declaration element to query.

Returns a definition that corresponds to the subtype of the object, as specified by a subtype_indication, subtype_mark, access_definition, or full type definition.

For a single_task_declaration or single_protected_declaration, returns the task_definition or protected_definition. If no task_definition is given explicitly (the reserved word is is not written), an empty task_definition is returned for which Is_Task_Definition_Present returns False..

If an empty task_definition E is returned, then Is_Part_of_Implicit(E) = False; Element_Span(E) returns a value where the First_Column_Number > Last_Column_Number and First_Line = Last_Line = the line of the semicolon; Element_Image(E) = ""; and Lines(E) returns a single line whose Line_Image = "".

For a Component_Declaration, returns the Component_Definition following the colon.

For all other object_declaration variables or constants, parameter or discriminant specifications, object renamings, or formal objects, returns the subtype_indication, subtype_mark, access_definition or array_type_definition following the colon.

Declaration expects an element that has one of the following Declaration_Kinds:

A_Variable_Declaration
A_Constant_Declaration
A_Deferred_Constant_Declaration
A_Single_Protected_Declaration
A_Single_Task_Declaration
A_Component_Declaration
A_Discriminant_Specification
A_Parameter_Specification
A_Formal_Object_Declaration
An_Object_Renaming_Declaration
A_Return_Object_Specification

Raises ASIS_Inappropriate_Element with a Status of Value_Error for any element that does not have one of these expected kinds.

Returns an element that has one of the following Definition_Kinds:

Not_A_Definition
A_Type_Definition that has one of the following Type_Kinds:
  A_Constrained_Array_Definition
A_Subtype_Indication
A_Task_Definition
A_Protected_Definition
A_Component_Definition
An_Access_Definition

NOTE Asis.Declarations.Object_Declaration_View, Asis.Declarations.Declaration_Subtype_Mark and the value An_Access_Definition_Trait of Trait_Kinds type are obsolescent and should not be used in new applications that are supposed to analyze Ada 2005 code.

15.10 function Initialization_Expression

```
function Initialization_Expression (Declaration : in Asis.Declaration)
                        return Asis.Expression;
```

Declaration specifies the object declaration to query.

Returns the initialization expression [:= expression] of the declaration.

Returns a Nil_Element if the declaration does not include an explicit initialization.

Declaration expects an element that has one of the following Declaration_Kinds:

A_Variable_Declaration
A_Constant_Declaration
An_Integer_Number_Declaration
A_Real_Number_Declaration
A_Discriminant_Specification
A_Component_Declaration
A_Parameter_Specification
A_Formal_Object_Declaration
A_Return_Object_Specification

Raises ASIS_Inappropriate_Element with a Status of Value_Error for any element that does not have one of these expected kinds.

Returns an element that has one of the following Element_Kinds:

Not_An_Element
An_Expression

15.11 function Corresponding_Constant_Declaration

```
function Corresponding_Constant_Declaration
            (Name : in Asis.Defining_Name)
           return Asis.Declaration;
```

Name specifies the name of a constant declaration to query.

Returns the corresponding full constant declaration when given the name from a deferred constant declaration.

Returns the corresponding deferred constant declaration when given the name from a full constant declaration.

Returns A_Pragma if the deferred constant declaration is completed by pragma Import.

Returns a Nil_Element if the full constant declaration has no corresponding deferred constant declaration.

Raises ASIS_Inappropriate_Element with a Status of Value_Error if the argument is not the name of a constant or a deferred constant.

The name of a constant declaration is available from both the Names and the Corresponding_Name_Definition queries.

Name expects an element that has the following Element_Kinds:

A_Defining_Name that has one of the following Declaration_Kinds:
Not_A_Declaration
A_Constant_Declaration
A_Deferred_Constant_Declaration

Raises ASIS_Inappropriate_Element with a Status of Value_Error for any element that does not have one of these expected kinds.

Returns an element that has one of the following Element_Kinds:

Not_An_Element
A_Declaration
A_Pragma

15.12 function Corresponding_Type_Declaration

```
function Corresponding_Type_Declaration
            (Declaration : in Asis.Declaration) return Asis.Declaration;

function Corresponding_Type_Declaration
            (Declaration : in Asis.Declaration;
             The_Context : in Asis.Context) return Asis.Declaration;
```

Declaration specifies the type declaration to query. The_Context specifies the program Context to use for obtaining package body information.

Returns the corresponding full type declaration when given a private or incomplete type declaration. Returns the corresponding private or incomplete type declaration when given a full type declaration.

These two function calls will always produce identical results:

```
Decl2 := Corresponding_Type_Declaration (Decl1);
Decl2 := Corresponding_Type_Declaration
            (Decl1,
             Enclosing_Context (Enclosing_Compilation_Unit (Decl1)));
```

Returns a Nil_Element when a full type declaration is given that has no corresponding private or incomplete type declaration, or when a corresponding type declaration does not exist within The_Context.

If The_Context specifies a context that is different from the Enclosing_Context(Enclosing_Compilation_Unit(Declaration)), returns the corresponding type declaration for the element in The_Context that Is_Equal to Declaration. If no such element exists in The_Context that Is_Equal to Declaration, returns A_Nil_Element.

Declaration expects an element that has one of the following Declaration_Kinds:

An_Ordinary_Type_Declaration
A_Task_Type_Declaration
A_Protected_Type_Declaration
An_Incomplete_Type_Declaration
A_Private_Type_Declaration
A_Private_Extension_Declaration

Raises ASIS_Inappropriate_Element with a Status of Value_Error for any element that does not have one of these expected kinds.

Returns an element that has one of the following Declaration_Kinds:

Not_A_Declaration
An_Ordinary_Type_Declaration
A_Task_Type_Declaration
A_Protected_Type_Declaration
An_Incomplete_Type_Declaration
A_Private_Type_Declaration
A_Private_Extension_Declaration

15.13 function Corresponding_First_Subtype

```
function Corresponding_First_Subtype
              (Declaration : in Asis.Declaration)
                  return Asis.Declaration;
```

Declaration specifies the subtype_declaration to query.

This function recursively unwinds subtyping to return at a type_declaration that defines the first subtype of the argument.

Returns a declaration that Is_Identical to the argument if the argument is already the first subtype.

Declaration expects an element that has one of the following Declaration_Kinds:

An_Ordinary_Type_Declaration
A_Task_Type_Declaration
A_Protected_Type_Declaration
A_Private_Type_Declaration
A_Private_Extension_Declaration
A_Subtype_Declaration
A_Formal_Type_Declaration

Raises ASIS_Inappropriate_Element with a Status of Value_Error for any element that does not have one of these expected kinds.

Returns an element that has one of the following Declaration_Kinds:

An_Ordinary_Type_Declaration
A_Task_Type_Declaration
A_Protected_Type_Declaration
A_Private_Type_Declaration
A_Private_Extension_Declaration
A_Formal_Type_Declaration

15.14 function Corresponding_Last_Constraint

```
function Corresponding_Last_Constraint
              (Declaration : in Asis.Declaration)
                  return Asis.Declaration;
```

Declaration specifies the subtype_declaration or type_declaration to query.

This function recursively unwinds subtyping to return at a declaration that is either a type_declaration or subtype_declaration that imposes an explicit constraint on the argument.

Unwinds a minimum of one level of subtyping even if an argument declaration itself has a constraint.

Returns a declaration that Is_Identical to the argument if the argument is a type_declaration, i.e. the first subtype.

Declaration expects an element that has one of the following Declaration_Kinds:

An_Ordinary_Type_Declaration
A_Task_Type_Declaration
A_Protected_Type_Declaration
A_Private_Type_Declaration
A_Private_Extension_Declaration
A_Subtype_Declaration

A_Formal_Type_Declaration

Raises ASIS_Inappropriate_Element with a Status of Value_Error for any element that does not have one of these expected kinds.

Returns an element that has one of the following Declaration_Kinds:

An_Ordinary_Type_Declaration
A_Task_Type_Declaration
A_Protected_Type_Declaration
A_Private_Type_Declaration
A_Private_Extension_Declaration
A_Subtype_Declaration
A_Formal_Type_Declaration

15.15 function Corresponding_Last_Subtype

```
function Corresponding_Last_Subtype
              (Declaration : in Asis.Declaration)
                  return Asis.Declaration;
```

Declaration specifies the subtype_declaration or type_declaration to query.

This function unwinds subtyping a single level to arrive at a declaration that is either a type_declaration or subtype_declaration.

Returns a declaration that Is_Identical to the argument if the argument is a type_declaration (i.e., the first subtype).

Declaration expects an element that has one of the following Declaration_Kinds:

An_Ordinary_Type_Declaration
A_Task_Type_Declaration
A_Protected_Type_Declaration
A_Private_Type_Declaration
A_Private_Extension_Declaration
A_Subtype_Declaration
A_Formal_Type_Declaration

Raises ASIS_Inappropriate_Element with a Status of Value_Error for any element that does not have one of these expected kinds.

Returns an element that has one of the following Declaration_Kinds:

An_Ordinary_Type_Declaration
A_Task_Type_Declaration
A_Protected_Type_Declaration
A_Private_Type_Declaration
A_Private_Extension_Declaration
A_Subtype_Declaration
A_Formal_Type_Declaration

15.16 function Corresponding_Aspect_Clauses

```
function Corresponding_Aspect_Clauses
            (Defining_Name : in Asis.Defining_Name)
            return Asis.Aspect_Clause_List;
```

Defining_Name specifies the defining_name to query.

Returns all aspect_clause elements that apply to the named entity.

Returns a Nil_Element_List if no clauses apply to the named entity.

The clauses returned may be the clauses applying to a parent type if the name refers to a type which is a derived type with no explicit aspect clause. These clauses are not Is_Part_Of_Implicit, they are the An_Aspect_Clause elements specified in conjunction with the declaration of the parent type.

Defining_Name expects an element of appropriate Element_Kinds:

A_Defining_Name

Raises ASIS_Inappropriate_Element with a Status of Value_Error if Defining_Name does not have this expected kind.

Returns a list of elements that each have one of the following Clause_Kinds:

An_Aspect_Clause

15.17 function Specification_Subtype_Definition

```
function Specification_Subtype_Definition
          (Specification : in Asis.Declaration)
              return Asis.Discrete_Subtype_Definition;
```

Specification specifies the loop_parameter_specification or Entry_Index_Specification to query.

Returns the Discrete_Subtype_Definition of the specification.

Specification expects an element that has one of the following Declaration_Kinds:

A_Loop_Parameter_Specification
An_Entry_Index_Specification

Raises ASIS_Inappropriate_Element with a Status of Value_Error for any element that does not have one of these expected kinds.

Returns an element that has the following Definition_Kinds:

A_Discrete_Subtype_Definition

15.18 function Parameter_Profile

```
function Parameter_Profile (Declaration : in Asis.Declaration)
                           return Asis.Parameter_Specification_List;
```

Declaration specifies the subprogram or entry declaration to query.

Returns a list of parameter specifications in the formal part of the subprogram or entry declaration, in their order of appearance.

Returns a Nil_Element_List if the subprogram or entry has no parameters.

Results of this query may vary across ASIS implementations. Some implementations normalize all multiple name parameter specifications into an equivalent sequence of corresponding single name parameter specifications. See Ada Standard 3.3.1(7).

Declaration expects an element that has one of the following Declaration_Kinds:

A_Procedure_Declaration
A_Function_Declaration
A_Procedure_Body_Declaration
A_Function_Body_Declaration
A_Procedure_Renaming_Declaration
A_Function_Renaming_Declaration
An_Entry_Declaration
An_Entry_Body_Declaration

A_Procedure_Body_Stub
A_Function_Body_Stub
A_Generic_Function_Declaration
A_Generic_Procedure_Declaration
A_Formal_Function_Declaration
A_Formal_Procedure_Declaration

Raises ASIS_Inappropriate_Element with a Status of Value_Error for any element that does not have one of these expected kinds.

Returns a list of elements that have the following Declaration_Kinds:

A_Parameter_Specification

15.19 function Result_Subtype

```
function Result_Subtype (Declaration : in Asis.Declaration)
                         return Asis.Definition;
```

Declaration specifies the function declaration to query.

Returns a definition that corresponds to the result subtype of the function, as specified by a subtype_indication (with no specified constraint) or an access_definition.

Declaration expects an element of Declaration_Kinds:

A_Function_Declaration
A_Function_Body_Declaration
A_Function_Body_Stub
A_Function_Renaming_Declaration
A_Generic_Function_Declaration
A_Formal_Function_Declaration

Raises ASIS_Inappropriate_Element with a Status of Value_Error for any element that does not have one of these expected kinds.

Returns an element that has one of the following Definition_Kinds:

A_Subtype_Indication
An_Access_Definition

15.20 function Body_Declarative_Items

```
function Body_Declarative_Items (Declaration : in Asis.Declaration;
                                 Include_Pragmas : in Boolean := False)
                                return Asis.Element_List;
```

Declaration specifies the body declaration to query. Include_Pragmas specifies whether pragmas are to be returned.

Returns a list of all basic declarations, representation and operational items, use clauses, and pragmas in the declarative part of the body, in their order of appearance.

Returns a Nil_Element_List if there are no declarative_item or pragma elements.

Results of this query may vary across ASIS implementations. Some implementations normalize all multi-name declarations into an equivalent sequence of corresponding single name object declarations. See Ada Standard 3.3.1(7).

Declaration expects an element that has one of the following Declaration_Kinds:

A_Function_Body_Declaration
A_Procedure_Body_Declaration
A_Package_Body_Declaration

A_Task_Body_Declaration
An_Entry_Body_Declaration

Raises ASIS_Inappropriate_Element with a Status of Value_Error for any element that does not have one of these expected kinds.

Returns a list of elements that have one of the following Element_Kinds:

A_Pragma
A_Declaration
A_Clause

15.21 function Body_Statements

```
function Body_Statements (Declaration : in Asis.Declaration;
                          Include_Pragmas : in Boolean := False)
                         return Asis.Statement_List;
```

Declaration specifies the body declaration to query. Include_Pragmas specifies whether pragmas are to be returned.

Returns a list of the statements and pragmas for the body, in their order of appearance.

Returns a Nil_Element_List if there are no statements or pragmas.

Declaration expects an element that has one of the following Declaration_Kinds:

A_Function_Body_Declaration
A_Procedure_Body_Declaration
A_Package_Body_Declaration
A_Task_Body_Declaration
An_Entry_Body_Declaration

Raises ASIS_Inappropriate_Element with a Status of Value_Error for any element that does not have one of these expected kinds.

Returns a list of elements that each have one of the following Element_Kinds:

A_Pragma
A_Statement

15.22 function Body_Exception_Handlers

```
function Body_Exception_Handlers (Declaration : in Asis.Declaration;
                                  Include_Pragmas : in Boolean := False)
                                 return Asis.Exception_Handler_List;
```

Declaration specifies the body declaration to query. Include_Pragmas specifies whether pragmas are to be returned.

Returns a list of the exception_handler elements of the body, in their order of appearance.

The only pragmas returned are those following the reserved word **exception** and preceding the reserved word **when** of the first exception handler.

Returns a Nil_Element_List if there are no exception_handler or pragma elements.

Declaration expects an element that has one of the following Declaration_Kinds:

A_Function_Body_Declaration
A_Procedure_Body_Declaration
A_Package_Body_Declaration
A_Task_Body_Declaration

An_Entry_Body_Declaration

Raises ASIS_Inappropriate_Element with a Status of Value_Error for any element that does not have one of these expected kinds.

Returns a list of elements that each have one of the following Element_Kinds:

An_Exception_Handler
A_Pragma

15.23 function Is_Name_Repeated (declaration)

```
function Is_Name_Repeated
            (Declaration : in Asis.Declaration) return Boolean;
```

Declaration specifies the declaration to query.

Returns True if the name of the declaration is repeated after the "end" which terminates the declaration.

Returns False for any unexpected Element.

Declaration expects an element that has one of the following Declaration_Kinds:

A_Package_Declaration
A_Package_Body_Declaration
A_Procedure_Body_Declaration
A_Function_Body_Declaration
A_Generic_Package_Declaration
A_Task_Type_Declaration
A_Single_Task_Declaration
A_Task_Body_Declaration
A_Protected_Type_Declaration
A_Single_Protected_Declaration
A_Protected_Body_Declaration
An_Entry_Body_Declaration

15.24 function Corresponding_Declaration (declaration)

```
function Corresponding_Declaration
            (Declaration : in Asis.Declaration)
            return Asis.Declaration;

function Corresponding_Declaration
            (Declaration : in Asis.Declaration;
             The_Context : in Asis.Context)
            return Asis.Declaration;
```

Declaration specifies the specification to query. The_Context specifies a Context to use.

Returns the corresponding specification of a subprogram, package, or task body declaration. Returns the expanded generic specification template for generic instantiations. The argument can be a Unit_Declaration from a Compilation_Unit, or, it can be any appropriate body declaration from any declarative context.

These two function calls will always produce identical results:

```
Decl2 := Corresponding_Declaration (Decl1);
Decl2 := Corresponding_Declaration
            (Decl1, Enclosing_Context (Enclosing_Compilation_Unit (Decl1)));
```

If a specification declaration is given, the same element is returned, unless it is a generic instantiation or an inherited subprogram declaration (see below).

If a subprogram renaming declaration is given:

a) in case of renaming-as-declaration, the same element is returned;

b) in case of renaming-as-body, the subprogram declaration completed by this subprogram renaming declaration is returned. (Ada Standard, 8.5.4(1))

Returns a Nil_Element if no explicit specification exists, or the declaration is the proper body of a subunit.

The parameter The_Context is used to locate the corresponding specification within a particular Context. The_Context need not be the Enclosing_Context of the Declaration. Any non-Nil result will always have The_Context as its Enclosing_Context. This implies that while a non-Nil result may be Is_Equal with the argument, it will only be Is_Identical if the Enclosing_Context of the Declaration is the same as the parameter The_Context.

If The_Context specifies a context that is different from the Enclosing_Context(Enclosing_Compilation_Unit(Declaration)), returns the declaration for the element in The_Context that Is_Equal to Declaration. If no such element exists in The_Context that Is_Equal to Declaration, returns A_Nil_Element.

If a generic instantiation is given, the expanded generic specification template representing the instance is returned and Is_Part_Of_Instance. For example, an argument that is A_Package_Instantiation, results in a value that is A_Package_Declaration that can be analyzed with all appropriate queries.

Returns the declaration of the generic child unit corresponding to an implicit generic child unit specification. Ada Standard 10.1.1(19).

The Enclosing_Element of the expanded specification is the generic instantiation. The Enclosing_Compilation_Unit of the expanded template is that of the instantiation.

If an inherited subprogram declaration is given, the specification returned is the one for the user-defined subprogram from which the argument was ultimately inherited.

Declaration expects an element that has one of the following Declaration_Kinds for returning a specification:

A_Function_Body_Declaration
A_Function_Renaming_Declaration (renaming-as-body)
A_Function_Body_Stub
A_Function_Instantiation
A_Package_Body_Declaration
A_Package_Body_Stub
A_Package_Instantiation
A_Procedure_Body_Declaration
A_Procedure_Renaming_Declaration (renaming-as-body)
A_Procedure_Body_Stub
A_Procedure_Instantiation
A_Task_Body_Declaration
A_Task_Body_Stub
A_Protected_Body_Declaration
A_Protected_Body_Stub
A_Formal_Package_Declaration
A_Formal_Package_Declaration_With_Box
An_Entry_Body_Declaration

Declaration expects an element that has one of the following Declaration_Kinds for returning the argument Declaration:

A_Function_Declaration

A_Function_Renaming_Declaration (renaming-as-declaration)
A_Generic_Function_Declaration
A_Generic_Package_Declaration
A_Generic_Procedure_Declaration
A_Package_Declaration
A_Package_Renaming_Declaration
A_Procedure_Declaration
A_Procedure_Renaming_Declaration (renaming-as-declaration)
A_Single_Task_Declaration
A_Task_Type_Declaration
A_Protected_Type_Declaration
A_Single_Protected_Declaration
A_Generic_Package_Renaming_Declaration
A_Generic_Procedure_Renaming_Declaration
A_Generic_Function_Renaming_Declaration
An_Entry_Declaration

Raises ASIS_Inappropriate_Element with a Status of Value_Error for any element that does not have one of these expected kinds.

Returns an element that has one of the following Declaration_Kinds:

Not_A_Declaration
A_Function_Declaration
A_Function_Renaming_Declaration
A_Generic_Function_Declaration
A_Generic_Package_Declaration
A_Generic_Procedure_Declaration
A_Package_Declaration
A_Package_Renaming_Declaration
A_Procedure_Declaration
A_Procedure_Renaming_Declaration
A_Single_Task_Declaration
A_Task_Type_Declaration
A_Protected_Type_Declaration
A_Single_Protected_Declaration
An_Entry_Declaration

15.25 function Corresponding_Body (declaration)

```
function Corresponding_Body (Declaration : in Asis.Declaration)
                            return Asis.Declaration;

function Corresponding_Body (Declaration : in Asis.Declaration;
                             The_Context : in Asis.Context)
                             return Asis.Declaration;
```

Declaration specifies the specification to query. The_Context specifies a Context to use.

Returns the corresponding body for a given subprogram, package, or task specification declaration. Returns the expanded generic body template for generic instantiations. The argument can be a Unit_Declaration from a Compilation_Unit, or, it can be any appropriate specification declaration from any declarative context.

These two function calls will always produce identical results:

```
Decl2 := Corresponding_Body (Decl1);
Decl2 := Corresponding_Body
         (Decl1, Enclosing_Context (Enclosing_Compilation_Unit (Decl1)));
```

If a body declaration is given, the same element is returned.

Returns a Nil_Element if no body exists in The_Context.

The parameter The_Context is used to locate the corresponding specification within a particular Context. The_Context need not be the Enclosing_Context of the Declaration. Any non-Nil result will always have The_Context as its Enclosing_Context. This implies that while a non-Nil result may be Is_Equal with the argument, it will only be Is_Identical if the Enclosing_Context of the Declaration is the same as the parameter The_Context.

If The_Context specifies a context that is different from the Enclosing_Context(Enclosing_Compilation_Unit(Declaration)), returns the declaration for the element in The_Context that Is_Equal to Declaration. If no such element exists in The_Context that Is_Equal to Declaration, returns A_Nil_Element.

Implicit predefined operations (e.g., "+", "=", etc.) will not typically have unit bodies. (Corresponding_Body returns a Nil_Element.) User-defined overloads of the predefined operations will have Corresponding_Body values once the bodies have inserted into the environment. The Corresponding_Body of an inherited subprogram is that of the original user-defined subprogram.

If a generic instantiation is given, the body representing the expanded generic body template is returned. (i.e., an argument that is A_Package_Instantiation, results in a value that is A_Package_Body_Declaration that can be analyzed with all appropriate ASIS queries).

Returns a Nil_Element if the body of the generic has not yet been compiled or inserted into the Ada Environment Context.

The Enclosing_Element of the expanded body is the generic instantiation. The Enclosing_Compilation_Unit of the expanded template is that of the instantiation.

Returns Nil_Element for an implicit generic child unit specification. Ada Standard 10.1.1(19).

Returns A_Pragma if the Declaration is completed by pragma Import.

Returns A_Nil_Element for a null procedure or an abstract procedure.

Declaration expects an element that has one of the following Declaration_Kinds returning a body:

A_Function_Declaration
A_Function_Instantiation
A_Generic_Package_Declaration
A_Generic_Procedure_Declaration
A_Generic_Function_Declaration
A_Package_Declaration
A_Package_Instantiation
A_Procedure_Declaration
A_Procedure_Instantiation
A_Single_Task_Declaration
A_Task_Type_Declaration
A_Protected_Type_Declaration
A_Single_Protected_Declaration
A_Formal_Package_Declaration
A_Formal_Package_Declaration_With_Box
An_Entry_Declaration (restricted to protected entry)

Declaration expects an element that has one of the following Declaration_Kinds for returning the argument Declaration:

A_Function_Body_Declaration
A_Function_Body_Stub
A_Function_Renaming_Declaration
A_Package_Body_Declaration
A_Package_Body_Stub
A_Package_Renaming_Declaration
A_Procedure_Body_Declaration
A_Procedure_Renaming_Declaration
A_Procedure_Body_Stub
A_Task_Body_Declaration
A_Task_Body_Stub
A_Protected_Body_Declaration
A_Protected_Body_Stub
A_Generic_Package_Renaming_Declaration
A_Generic_Procedure_Renaming_Declaration
A_Generic_Function_Renaming_Declaration
An_Entry_Body_Declaration

Returns an element that has one of the following Element_Kinds:

Not_An_Element
A_Pragma
A_Declaration that has one of Declaration_Kinds:
  Not_A_Declaration
  A_Function_Body_Declaration
  A_Function_Body_Stub
  A_Function_Renaming_Declaration
  A_Package_Body_Declaration
  A_Package_Body_Stub
  A_Procedure_Body_Declaration
  A_Procedure_Renaming_Declaration
  A_Procedure_Body_Stub
  A_Task_Body_Declaration
  A_Task_Body_Stub
  A_Protected_Body_Declaration
  A_Protected_Body_Stub
  An_Entry_Body_Declaration

15.26 function Corresponding_Subprogram_Derivation

```
function Corresponding_Subprogram_Derivation
          (Declaration : in Asis.Declaration)
              return Asis.Declaration;
```

Declaration specifies an implicit inherited subprogram declaration.

Returns the subprogram declaration from which the given implicit inherited subprogram argument was inherited. The result can itself be an implicitly inherited subprogram.

Declaration expects an element that has one of the following Declaration_Kinds:

A_Function_Declaration
A_Procedure_Declaration

Raises ASIS_Inappropriate_Element with a Status of Value_Error for any element that does not have one of these expected kinds.

Returns an element that has one of the following Declaration_Kinds:

A_Function_Body_Declaration
A_Function_Declaration
A_Function_Renaming_Declaration
A_Procedure_Body_Declaration
A_Procedure_Declaration
A_Procedure_Renaming_Declaration

Raises ASIS_Inappropriate_Element for a subprogram declaration that is not Is_Part_Of_Inherited.

15.27 function Corresponding_Type

```
function Corresponding_Type (Declaration : in Asis.Declaration)
                            return Asis.Type_Definition;
```

Declaration specifies the subprogram_declaration to query.

Returns the type definition for which this entity is an implicit declaration. The result will often be a derived type. However, this query also works for declarations of predefined operators such as "+" and "=". Raises ASIS_Inappropriate_Element if the argument is not an implicit declaration resulting from the declaration of a type.

Declaration expects an element that has one of the following Declaration_Kinds:

A_Function_Declaration
A_Procedure_Declaration

Raises ASIS_Inappropriate_Element with a Status of Value_Error for any element that does not have one of these expected kinds.

Returns an element that has one of the following Definition_Kinds:

A_Type_Definition
A_Formal_Type_Definition

15.28 function Corresponding_Equality_Operator

```
function Corresponding_Equality_Operator
         (Declaration : in Asis.Declaration) return Asis.Declaration;
```

Declaration specifies an equality or an inequality operator declaration.

If given an explicit Declaration of "=" whose result type is Boolean:

- Returns the complementary implicit "/=" operator declaration.
- Returns a Nil_Element if the Ada implementation has not defined an implicit "/=" for the "=". Implementations of this sort will transform a A/=B expression into a **not**(A=B) expression. The function call representing the **not** operation is Is_Part_Of_Implicit in this case.

If given an implicit declaration of "/=" whose result type is Boolean:

- Returns the complementary explicit "=" operator declaration.

Returns a Nil_Element for any other function declaration.

Declaration expects an element that has the following Declaration_Kinds:

A_Function_Declaration

Raises ASIS_Inappropriate_Element with a Status of Value_Error for any element that does not have one of these expected kinds.

Returns an element that has the following Declaration_Kinds:

A_Function_Declaration

15.29 function Visible_Part_Declarative_Items

```
function Visible_Part_Declarative_Items
            (Declaration     : in Asis.Declaration;
             Include_Pragmas : in Boolean := False)
            return Asis.Declarative_Item_List;
```

Declaration specifies the package to query. Include_Pragmas specifies whether pragmas are to be returned.

Returns a list of all basic declarations, representation and operational items, use clauses, and pragmas in the visible part of a package, in their order of appearance.

Results of this query may vary across ASIS implementations. Some implementations normalize all multi-name object declarations into an equivalent sequence of corresponding single name object declarations. See Ada Standard 3.3.1(7).

Declaration expects an element that has one of the following Declaration_Kinds:

A_Generic_Package_Declaration
A_Package_Declaration

Raises ASIS_Inappropriate_Element with a Status of Value_Error for any element that does not have one of these expected kinds.

Returns a list of elements that each have one of the following Element_Kinds:

A_Declaration
A_Pragma
A_Clause

15.30 function Is_Private_Present (declaration)

```
function Is_Private_Present
             (Declaration : in Asis.Declaration) return Boolean;
```

Declaration specifies the declaration to query.

Returns True if the argument is a package specification which has a reserved word **private** which marks the beginning of a (possibly empty) private part.

Returns False for any package specification without a private part. Returns False for any unexpected Element.

Declaration expects an element that has one of the following Declaration_Kinds:

A_Generic_Package_Declaration
A_Package_Declaration

Raises ASIS_Inappropriate_Element with a Status of Value_Error for any element that does not have one of these expected kinds.

15.31 function Private_Part_Declarative_Items

```
function Private_Part_Declarative_Items
            (Declaration     : in Asis.Declaration;
             Include_Pragmas : in Boolean := False)
            return Asis.Declarative_Item_List;
```

Declaration specifies the package to query. Include_Pragmas specifies whether pragmas are to be returned.

Returns a list of all basic declarations, representation and operational items, use clauses, and pragmas in the private part of a package in their order of appearance.

Results of this query may vary across ASIS implementations. Some implementations normalize all multi-name object declarations into an equivalent sequence of corresponding single name object declarations. See Ada Standard 3.3.1(7).

Declaration expects an element that has one of the following Declaration_Kinds:

A_Generic_Package_Declaration
A_Package_Declaration

Raises ASIS_Inappropriate_Element with a Status of Value_Error for any element that does not have one of these expected kinds.

Returns a list of elements that each have one of the following Element_Kinds:

A_Declaration
A_Pragma
A_Clause

15.32 function Renamed_Entity

```
function Renamed_Entity (Declaration : in Asis.Declaration)
                         return Asis.Name;
```

Declaration specifies the rename declaration to query.

Returns the name expression that follows the reserved word **renames** in the renaming declaration.

Declaration expects an element that has one of the following Declaration_Kinds:

An_Exception_Renaming_Declaration
A_Function_Renaming_Declaration
An_Object_Renaming_Declaration
A_Package_Renaming_Declaration
A_Procedure_Renaming_Declaration
A_Generic_Package_Renaming_Declaration
A_Generic_Procedure_Renaming_Declaration
A_Generic_Function_Renaming_Declaration

Raises ASIS_Inappropriate_Element with a Status of Value_Error for any element that does not have one of these expected kinds.

Returns an element that has the following Element_Kinds:

An_Expression

15.33 function Corresponding_Base_Entity

```
function Corresponding_Base_Entity (Declaration : in Asis.Declaration)
                                    return Asis.Name;
```

Declaration specifies the renaming declaration to query.

The base entity is defined to be the renamed entity that is not itself defined by another renaming declaration.

If the name following the reserved word **renames** is itself declared by a previous renaming_declaration, then this query unwinds the renamings by recursively operating on the previous renaming_declaration.

Otherwise, the name following the reserved word **renames** is returned.

Declaration expects an element that has one of the following Declaration_Kinds:

An_Object_Renaming_Declaration
An_Exception_Renaming_Declaration
A_Procedure_Renaming_Declaration
A_Function_Renaming_Declaration
A_Package_Renaming_Declaration
A_Generic_Package_Renaming_Declaration
A_Generic_Procedure_Renaming_Declaration
A_Generic_Function_Renaming_Declaration

Raises ASIS_Inappropriate_Element with a Status of Value_Error for any element that does not have one of these expected kinds.

Returns an element that has the following Element_Kinds:

An_Expression

15.34 function Protected_Operation_Items

```
function Protected_Operation_Items
              (Declaration     : in Asis.Declaration;
               Include_Pragmas : in Boolean := False)
              return Asis.Declaration_List;
```

Declaration specifies the protected_body declaration to query. Include_Pragmas specifies whether pragmas are to be returned.

Returns a list of protected_operation_item and pragma elements of the protected_body, in order of appearance.

Returns a Nil_Element_List if there are no items or pragmas.

Declaration expects an element that has the following Declaration_Kinds:

A_Protected_Body_Declaration

Raises ASIS_Inappropriate_Element with a Status of Value_Error for any element that does not have one of these expected kinds.

Returns a list of elements that each have one of the following Element_Kinds:

A_Pragma
A_Declaration that has one of the following Declaration_Kinds:
  A_Procedure_Declaration
  A_Function_Declaration
  A_Procedure_Body_Declaration
  A_Function_Body_Declaration
  An_Entry_Body_Declaration
A_Clause that has one of Clause_Kinds:
  An_Aspect_Clause

15.35 function Entry_Family_Definition

```
function Entry_Family_Definition (Declaration : in Asis.Declaration)
                                 return Asis.Discrete_Subtype_Definition;
```

Declaration specifies the entry declaration to query.

Returns the Discrete_Subtype_Definition element for the entry family of an entry_declaration.

Returns a Nil_Element if the entry_declaration does not define a family of entries.

Declaration expects an element that has the following Declaration_Kinds:

An_Entry_Declaration

Raises ASIS_Inappropriate_Element with a Status of Value_Error for any element that does not have one of these expected kinds.

Returns an element that has one of the following Definition_Kinds:

Not_A_Definition
A_Discrete_Subtype_Definition

15.36 function Entry_Index_Specification

```
function Entry_Index_Specification (Declaration : in Asis.Declaration)
                                   return Asis.Declaration;
```

Declaration specifies the entry body declaration to query.

Returns the An_Entry_Index_Specification element of an entry body declaration.

Returns a Nil_Element if the entry does not declare any An_Entry_Index_Specification element.

Declaration expects an element that has the following Declaration_Kinds:

An_Entry_Body_Declaration

Raises ASIS_Inappropriate_Element with a Status of Value_Error for any element that does not have one of these expected kinds.

Returns an element that has one of the following Declaration_Kinds:

Not_A_Declaration
An_Entry_Index_Specification

15.37 function Entry_Barrier

```
function Entry_Barrier (Declaration : in Asis.Declaration)
                       return Asis.Expression;
```

Declaration specifies the entry body declaration to query.

Returns the expression following the reserved word **when** in an entry body declaration.

Declaration expects an element that has the following Declaration_Kinds:

An_Entry_Body_Declaration

Raises ASIS_Inappropriate_Element with a Status of Value_Error for any element that does not have one of these expected kinds.

Returns an element that has the following Element_Kinds:

An_Expression

15.38 function Corresponding_Subunit

```
function Corresponding_Subunit (Body_Stub : in Asis.Declaration)
                 return Asis.Declaration;

function Corresponding_Subunit (Body_Stub   : in Asis.Declaration;
                                The_Context : in Asis.Context)
                 return Asis.Declaration;
```

Body_Stub specifies the stub to query. The_Context specifies a Context to use to locate the subunit.

Returns the Unit_Declaration of the subunit compilation unit corresponding to the body stub.

Returns a Nil_Element if the subunit does not exist in The_Context.

These two function calls will always produce identical results:

```
Decl2 := Corresponding_Subunit (Decl1);
Decl2 := Corresponding_Subunit
         (Decl1, Enclosing_Context (Enclosing_Compilation_Unit (Decl1)));
```

The parameter The_Context is used to locate the corresponding subunit body. Any non-Nil result will always have The_Context as its Enclosing_Context.

If The_Context specifies a context that is different from the Enclosing_Context(Enclosing_Compilation_Unit(Body_Stub)), returns the unit for the element in The_Context that Is_Equal to Body_Stub. If no such element exists in The_Context that Is_Equal to Body_Stub, returns A_Nil_Element.

Body_Stub expects an element that has one of the following Declaration_Kinds:

A_Function_Body_Stub
A_Package_Body_Stub
A_Procedure_Body_Stub
A_Task_Body_Stub
A_Protected_Body_Stub

Raises ASIS_Inappropriate_Element with a Status of Value_Error for any element that does not have one of these expected kinds.

Returns an element that has one of the following Declaration_Kinds:

Not_A_Declaration
A_Function_Body_Declaration
A_Package_Body_Declaration
A_Procedure_Body_Declaration
A_Task_Body_Declaration
A_Protected_Body_Declaration

15.39 function Is_Subunit

```
function Is_Subunit (Declaration : in Asis.Declaration) return Boolean;
```

Declaration specifies the declaration to query.

Returns True if the declaration is the proper_body of a subunit, and returns False otherwise (including if Declaration has an unexpected Declaration_Kinds).

Equivalent to:

```
Declaration = Unit_Declaration (Enclosing_Compilation_Unit (Declaration))
and Unit_Kind (Enclosing_Compilation_Unit (Declaration)) in A_Subunit
```

Declaration expects an element that has one of the following Declaration_Kinds:

A_Procedure_Body_Declaration
A_Function_Body_Declaration
A_Package_Body_Declaration
A_Task_Body_Declaration
A_Protected_Body_Declaration

15.40 function Corresponding_Body_Stub

```
function Corresponding_Body_Stub (Subunit : in Asis.Declaration)
                  return Asis.Declaration;

function Corresponding_Body_Stub (Subunit     : in Asis.Declaration;
                                  The_Context : in Asis.Context)
                  return Asis.Declaration;
```

Subunit specifies the Is_Subunit declaration to query. The_Context specifies a Context to use to locate the parent unit.

Returns the body stub declaration located in the subunit's parent unit.

Returns a Nil_Element if the parent unit does not exist in The_Context.

These two function calls will always produce identical results:

```
Decl2 := Corresponding_Body_Stub (Decl1);
Decl2 := Corresponding_Body_Stub
         (Decl1, Enclosing_Context (Enclosing_Compilation_Unit (Decl1)));
```

The parameter The_Context is used to locate the corresponding parent body. Any non-Nil result will always have The_Context as its Enclosing_Context.

If The_Context specifies a context that is different from the Enclosing_Context(Enclosing_Compilation_Unit(Subunit)), returns the declaration for the element in The_Context that Is_Equal to Subunit. If no such element exists in The_Context that Is_Equal to Subunit, returns A_Nil_Element.

Subunit expects an element for which Is_Subunit(Declaration) is True and that has one of the following Declaration_Kinds:

A_Function_Body_Declaration
A_Package_Body_Declaration
A_Procedure_Body_Declaration
A_Task_Body_Declaration
A_Protected_Body_Declaration

Raises ASIS_Inappropriate_Element with a Status of Value_Error for any element that does not have one of these expected kinds.

Returns an element that has one of the following Declaration_Kinds:

Not_A_Declaration
A_Function_Body_Stub
A_Package_Body_Stub
A_Procedure_Body_Stub
A_Task_Body_Stub
A_Protected_Body_Stub

15.41 function Generic_Formal_Part (declaration)

```
function Generic_Formal_Part
            (Declaration     : in Asis.Declaration;
             Include_Pragmas : in Boolean := False)
            return Asis.Element_List;
```

Declaration specifies the generic declaration to query. Include_Pragmas specifies whether pragmas are to be returned.

Returns a list of generic formal parameter declarations, use clauses, and pragmas, in their order of appearance.

Results of this query may vary across ASIS implementations. Some implementations normalize all multi-name object declarations into an equivalent sequence of corresponding single name object declarations. See Ada Standard 3.3.1(7).

Declaration expects an element that has one of the following Declaration_Kinds:

A_Generic_Package_Declaration
A_Generic_Procedure_Declaration
A_Generic_Function_Declaration

Raises ASIS_Inappropriate_Element with a Status of Value_Error for any element that does not have one of these expected kinds.

Returns a a list of elements that each have one of the following Element_Kinds:

A_Pragma
A_Declaration
A_Clause

Returns an element that has one of the following Declaration_Kinds if A_Declaration:

A_Formal_Object_Declaration
A_Formal_Type_Declaration
A_Formal_Procedure_Declaration
A_Formal_Function_Declaration
A_Formal_Package_Declaration
A_Formal_Package_Declaration_With_Box

Returns an element that has one of the following Clause_Kinds if A_Clause:

A_Use_Package_Clause
A_Use_Type_Clause

15.42 function Generic_Unit_Name

```
function Generic_Unit_Name (Declaration : in Asis.Declaration)
                           return Asis.Name;
```

Declaration specifies the generic instantiation to query.

Returns the name following the reserved word **new** in the generic instantiation. The name denotes the generic package, generic procedure, or generic function that is the template for this generic instance.

Declaration expects an element that has one of the following Declaration_Kinds:

A_Function_Instantiation
A_Package_Instantiation
A_Procedure_Instantiation
A_Formal_Package_Declaration
A_Formal_Package_Declaration_With_Box

Raises ASIS_Inappropriate_Element with a Status of Value_Error for any element that does not have one of these expected kinds.

Returns an element that has one of the following Expression_Kinds:

An_Identifier
An_Operator_Symbol
A_Selected_Component

15.43 function Generic_Actual_Part

```
function Generic_Actual_Part (Declaration : in Asis.Declaration;
                              Normalized  : in Boolean := False)
                             return Asis.Association_List;
```

Declaration specifies the generic_instantiation to query. Normalized specifies whether the normalized form is desired.

Returns a list of the generic_association elements of the instantiation.

Returns a Nil_Element_List if there are no generic_association elements.

An unnormalized list contains only explicit associations ordered as they appear in the program text. Each unnormalized association has an optional generic_formal_parameter_selector_name and an explicit_generic_actual_parameter component.

A normalized list contains artificial associations representing all explicit and default associations. It has a length equal to the number of generic_formal_parameter_declaration elements of the generic_formal_part of the template. The order of normalized associations matches the order of the generic_formal_parameter_declaration elements.

Each normalized association represents a one-on-one mapping of a generic_formal_parameter_declaration to the explicit or default expression or name. A normalized association has:

- one A_Defining_Name component that denotes the generic_formal_parameter_declaration, and
- one An_Expression component that is either:
  - the explicit_generic_actual_parameter,
  - a default_expression, or
  - a default_name from the generic_formal_parameter_declaration or an implicit naming expression which denotes the actual subprogram selected at the place of instantiation for a formal subprogram having A_Box_Default.

Declaration expects an element that has one of the following Declaration_Kinds:

A_Function_Instantiation
A_Package_Instantiation
A_Procedure_Instantiation
A_Formal_Package_Declaration

Raises ASIS_Inappropriate_Element with a Status of Value_Error for any element that does not have one of these expected kinds.

Returns a list of elements that each have the following Association_Kinds:

A_Generic_Association

Implementation Requirements

Normalized associations are Is_Normalized and Is_Part_Of_Implicit. Normalized associations provided by default are Is_Defaulted_Association. Normalized associations are never Is_Equal to unnormalized associations.

15.44 function Formal_Subprogram_Default

```
function Formal_Subprogram_Default
            (Declaration : in Asis.Generic_Formal_Parameter)
          return Asis.Name;
```

Declaration specifies the generic formal subprogram declaration to query.

Returns the name appearing after the reserved word **is** in the given generic formal subprogram declaration.

Declaration expects an element that has one of the following Declaration_Kinds:

A_Formal_Function_Declaration
A_Formal_Procedure_Declaration

or Declaration expects an element that has the following Subprogram_Default_Kinds:

A_Name_Default

Raises ASIS_Inappropriate_Element with a Status of Value_Error for any element that does not have one of these expected kinds.

Returns an element that has the following Element_Kinds:

An_Expression

15.45 function Corresponding_Generic_Element

```
function Corresponding_Generic_Element (Reference : in Asis.Element)
                                        return Asis.Defining_Name;
```

Reference specifies an expression that references an entity declared within the implicit specification of a generic instantiation, or, specifies the defining name of such an entity.

Given a reference to some implicit entity, whose declaration occurs within an implicit generic instance, returns the corresponding entity name definition from the generic template used to create the generic instance. (Ada Standard 12.3 (16))

Returns the first A_Defining_Name, from the generic template, that corresponds to the entity referenced.

Returns a Nil_Element if the argument does not refer to an entity declared as a component of a generic package instantiation. The entity name can refer to an ordinary declaration, an inherited subprogram declaration, or a predefined operator declaration.

Reference expects an element that has one of the following Element_Kinds:

A_Defining_Name
An_Expression that has one of the following Expression_Kinds:
  An_Identifier
  An_Operator_Symbol
  A_Character_Literal
  An_Enumeration_Literal

Raises ASIS_Inappropriate_Element with a Status of Value_Error for any element that does not have one of these expected kinds.

Returns an element that has one of the following Expression_Kinds:

Not_An_Element
A_Defining_Name

15.46 function Is_Dispatching_Operation

```
function Is_Dispatching_Operation (Declaration : in Asis.Element)
                                   return Boolean;
```

Declaration specifies the declaration to query.

Returns True if the declaration is a primitive subprogram of a tagged type.

Returns False for any unexpected argument.

Declaration expects an element that has one of the following Element_Kinds:

A_Procedure_Declaration
A_Function_Declaration
A_Procedure_Renaming_Declaration
A_Function_Renaming_Declaration
A_Procedure_Instantiation
A_Function_Instantiation

15.47 function Progenitor_List (declaration)

```
function Progenitor_List
     (Declaration : in Asis.Definition)
      return Asis.Name_List;
```

Declaration specifies the declaration to query.

Returns a list of subtype marks making up the interface_list in the argument declaration, in their order of appearance. If Declaration has no progenitors, an empty list is returned.

Declaration expects an element that has one of the following Declaration_Kinds:

A_Private_Extension_Declaration
A_Task_Type_Declaration
A_Protected_Type_Declaration
A_Single_Task_Declaration
A_Single_Protected_Declaration

Raises ASIS_Inappropriate_Element with a Status of Value_Error for any element that does not have one of these expected kinds.

Returns a list of elements that each have one of the following Expression_Kinds:

An_Identifier
A_Selected_Component

15.48 function Overriding_Indicator_Kind

```
function Overriding_Indicator_Kind -- 8.3.1 (2)
   (Declaration : in Asis.Declaration)
        return Overriding_Indicator_Kinds;
```

Declaration specifies the subprogram declaration to query.

Returns the kind of Overriding_Indicator for the subprogram declaration.

Returns Not_An_Overriding_Indicator for any unexpected Element.

Declaration expects an element that has one of the following Declaration_Kinds:

A_Procedure_Declaration
A_Function_Declaration
A_Procedure_Body_Declaration

A_Function_Body_Declaration
A_Null_Procedure_Declaration
A_Procedure_Renaming_Declaration
A_Function_Renaming_Declaration
An_Entry_Declaration
A_Procedure_Body_Stub
A_Function_Body_Stub
A_Procedure_Instantiation
A_Function_Instantiation

Section 16: package Asis.Definitions

The library package Asis.Definitions shall exist. The package shall provide interfaces equivalent to those described in the following subclauses.

Asis.Definitions encapsulates a set of queries that operate on A_Definition and An_Association elements.

16.1 function Corresponding_Type_Operators

```
function Corresponding_Type_Operators
        (Type_Definition : in Asis.Type_Definition)
            return Asis.Declaration_List;
```

Type_Definition specifies the type to query.

Returns a list of operators. These include all predefined operators, and all user-defined operator overloads, that have been implicitly or explicitly declared for the type. (Ada Standard 7.3.1(2))

This list includes only operators appropriate for the type, from the set:

```
and or xor = /= < <= > >= + - & * / mod rem ** abs not
```

Returns a Nil_Element_List if there are no predefined or overloaded operators for the type.

Returns a Nil_Element_List if the implementation does not provide such implicit declarations.

The Enclosing_Element for each implicit declaration is the declaration (type or object) that declared the type.

If a user-defined equality operator has been defined, an Ada implementation has two choices when dealing with an instance of the "/=" operator. a) treat A/=B as **not**(A=B), b) implicitly create a "/=" operator. Implementations that take the second alternative will include this implicit inequality operation in their result. Implementations that choose the first alternative are encouraged to hide this choice beneath the ASIS interface and to "fake" an inequality operation. Failing that, the function call, representing the **not** operation, must have Is_Part_Of_Implicit = True so that an ASIS application can tell the difference between a user-specified **not**(A=B) and an implementation-specific A/=B transformation.

Type_Definition expects an element that has one of the following Definition_Kinds:

A_Type_Definition
A_Private_Type_Definition
A_Tagged_Private_Type_Definition
A_Private_Extension_Definition
A_Task_Definition
A_Protected_Definition
A_Formal_Type_Definition

Raises ASIS_Inappropriate_Element with a Status of Value_Error for any element that does not have one of these expected kinds.

Returns a list of elements that each have one of the following Declaration_Kinds:

A_Function_Declaration
A_Function_Body_Declaration
A_Function_Body_Stub
A_Function_Renaming_Declaration

A_Function_Instantiation
A_Formal_Function_Declaration

Implementation Permissions

The result may or may not include language defined operators that have been overridden by user-defined overloads. Operators that are totally hidden, in all contexts, by user-defined operators may be omitted from the list.

Some implementations do not represent all forms of implicit declarations such that elements representing them can be easily provided. An implementation can choose whether or not to construct and provide artificial declarations for implicitly declared elements.

16.2 function Parent_Subtype_Indication

```
function Parent_Subtype_Indication
              (Type_Definition : in Asis.Type_Definition)
                    return Asis.Subtype_Indication;
```

Type_Definition specifies the derived_type_definition to query.

Returns the parent_subtype_indication following the reserved word **new**.

Type_Definition expects an element that has one of the following Type_Kinds:

A_Derived_Type_Definition
A_Derived_Record_Extension_Definition

Raises ASIS_Inappropriate_Element with a Status of Value_Error for any element that does not have one of these expected kinds.

Returns an element that has the following Definition_Kinds:

A_Subtype_Indication

16.3 function Record_Definition

```
function Record_Definition (Type_Definition : in Asis.Type_Definition)
                           return Asis.Definition;
```

Type_Definition specifies the definition to query.

Returns the record definition of the type_definition.

Type_Definition expects an element that has one of the following Type_Kinds:

A_Derived_Record_Extension_Definition
A_Record_Type_Definition
A_Tagged_Record_Type_Definition

Raises ASIS_Inappropriate_Element with a Status of Value_Error for any element that does not have one of these expected kinds.

Returns an element that has one of the following Definition_Kinds:

A_Record_Definition
A_Null_Record_Definition

16.4 function Implicit_Inherited_Declarations

```
function Implicit_Inherited_Declarations
              (Definition : in Asis.Definition)
                  return Asis.Declaration_List;
```

Definition specifies the derived type to query.

Returns a list of Is_Part_Of_Implicit inherited enumeration literals, discriminants, components, protected subprograms, or entries of a derived_type_definition whose parent type is an enumeration type, or a composite type other than an array type. See Ada Standard 3.4(10-14).

Returns a Nil_Element_List if the root type of derived_type_definition is not an enumeration, record, task, or protected type.

The Enclosing_Element for each of the implicit declarations is the Declaration argument.

Definition expects an element that has the following Definition_Kinds:

A_Type_Definition that has one of the following Type_Kinds:
  A_Derived_Type_Definition
  A_Derived_Record_Extension_Definition
A_Private_Extension_Definition
A_Formal_Type_Definition that has the following Formal_Type_Kinds:
  A_Formal_Derived_Type_Definition

Returns a list of elements that each have have one of the following Declaration_Kinds:

An_Enumeration_Literal_Specification
A_Discriminant_Specification
A_Component_Declaration
A_Procedure_Declaration
A_Function_Declaration
An_Entry_Declaration

16.5 function Implicit_Inherited_Subprograms

```
function Implicit_Inherited_Subprograms
            (Definition : in Asis.Definition)
            return Asis.Declaration_List;
```

Definition specifies the derived type to query.

Returns the list of user-defined inherited primitive subprograms that have been implicitly declared for the derived_type_definition.

The list result does not include hidden inherited subprograms (Ada Standard 8.3).

Returns a Nil_Element_List if there are no inherited subprograms for the derived type.

The Enclosing_Element for each of the subprogram declarations is the Definition argument.

Definition expects an element that has one of the following Definition_Kinds:

A_Type_Definition that has one of the following Type_Kinds:
  A_Derived_Type_Definition
  A_Derived_Record_Extension_Definition
A_Private_Extension_Definition
A_Formal_Type_Definition that has the following Formal_Type_Kinds:
  A_Formal_Derived_Type_Definition

Raises ASIS_Inappropriate_Element with a Status of Value_Error for any element that does not have one of these expected kinds.

Returns a list of elements that each have one of the following Declaration_Kinds:

A_Function_Declaration
A_Procedure_Declaration

16.6 function Corresponding_Root_Type

```
function Corresponding_Root_Type
                 (Type_Definition : in Asis.Type_Definition)
                  return Asis.Declaration;
```

Type_Definition specifies the derived_type_definition to query.

This function recursively unwinds all type derivations and subtyping to arrive at a full_type_declaration that is neither a derived type nor a subtype.

In case of numeric types, this function always returns some user-defined type, not an implicitly defined root type corresponding to A_Root_Type_Definition. The only ways to get implicitly declared numeric root or universal types are to ask for the type of a universal expression or from the parameter and result profile of a predefined operation working with numeric types.

Type_Definition expects an element that has one of the following Type_Kinds:

A_Derived_Type_Definition
A_Derived_Record_Extension_Definition

Raises ASIS_Inappropriate_Element with a Status of Value_Error for any element that does not have one of these expected kinds.

Returns an element that has one of the following Declaration_Kinds:

An_Ordinary_Type_Declaration
A_Task_Type_Declaration
A_Protected_Type_Declaration
A_Formal_Type_Declaration
A_Private_Type_Declaration
A_Private_Extension_Declaration

16.7 function Corresponding_Type_Structure

```
function Corresponding_Type_Structure
              (Type_Definition : in Asis.Type_Definition)
                  return Asis.Declaration;
```

Type_Definition specifies the derived_type_definition to query.

Returns the type structure from which the specified type definition has been derived. This function will recursively unwind derivations and subtyping until the type_declaration is a derived type whose parent type has a different representation or is not a derived type. See Ada Standard 13.6.

Type_Definition expects an element that has one of the following Type_Kinds:

A_Derived_Type_Definition
A_Derived_Record_Extension_Definition

Raises ASIS_Inappropriate_Element with a Status of Value_Error for any element that does not have one of these expected kinds.

Returns an element that has one of the following Declaration_Kinds:

An_Ordinary_Type_Declaration
A_Task_Type_Declaration
A_Protected_Type_Declaration
A_Formal_Type_Declaration
A_Private_Type_Declaration
A_Private_Extension_Declaration

16.8 function Enumeration_Literal_Declarations

```
function Enumeration_Literal_Declarations
           (Type_Definition : in Asis.Type_Definition)
            return Asis.Declaration_List;
```

Type_Definition specifies the enumeration type definition to query.

Returns a list of the literals declared in an enumeration_type_definition, in their order of appearance.

Type_Definition expects an element that has the following Type_Kinds:

An_Enumeration_Type_Definition

Raises ASIS_Inappropriate_Element with a Status of Value_Error for any element that does not have one of these expected kinds.

Returns a list of elements that each have the following Declaration_Kinds:

An_Enumeration_Literal_Specification

16.9 function Integer_Constraint

```
function Integer_Constraint
            (Type_Definition : in Asis.Type_Definition)
            return Asis.Range_Constraint;
```

Type_Definition specifies the signed_integer_type_definition to query.

Returns the range_constraint of the signed_integer_type_definition.

Type_Definition expects an element that has the following Type_Kinds:

A_Signed_Integer_Type_Definition

Raises ASIS_Inappropriate_Element with a Status of Value_Error for any element that does not have one of these expected kinds.

Returns an element that has the following Constraint_Kinds:

A_Simple_Expression_Range

16.10 function Mod_Static_Expression

```
function Mod_Static_Expression
            (Type_Definition : in Asis.Type_Definition)
            return Asis.Expression;
```

Type_Definition specifies the modular_type_definition to query.

Returns the static_expression following the reserved word **mod**.

Type_Definition expects an element that has the following Type_Kinds:

A_Modular_Type_Definition

Raises ASIS_Inappropriate_Element with a Status of Value_Error for any element that does not have one of these expected kinds.

Returns an element that has the following Element_Kinds:

An_Expression

16.11 function Digits_Expression

```
function Digits_Expression (Definition : in Asis.Definition)
                           return Asis.Expression;
```

Definition specifies the definition to query.

Returns the static_expression following the reserved word **digits**.

Definition expects an element that has one of the following Definition_Kinds:

A_Constraint that has the following Constraint_Kinds:
  A_Digits_Constraint
A_Type_Definition that has one of the following Type_Kinds:
  A_Floating_Point_Definition
  A_Decimal_Fixed_Point_Definition

Raises ASIS_Inappropriate_Element with a Status of Value_Error for any element that does not have one of these expected kinds.

Returns an element that has the following Element_Kinds:

An_Expression

16.12 function Delta_Expression

```
function Delta_Expression (Definition : in Asis.Definition)
                          return Asis.Expression;
```

Definition specifies the definition to query.

Returns the static_expression following the reserved word **delta**.

Definition expects an element that has one of the following Definition_Kinds:

A_Constraint that has the following Constraint_Kinds:
  A_Delta_Constraint
A_Type_Definition that has one of the following Type_Kinds:
  An_Ordinary_Fixed_Point_Definition
  A_Decimal_Fixed_Point_Definition

Raises ASIS_Inappropriate_Element with a Status of Value_Error for any element that does not have one of these expected kinds.

Returns an element that has the following Element_Kinds:

An_Expression

16.13 function Real_Range_Constraint

```
function Real_Range_Constraint
         (Definition : in Asis.Definition) return Asis.Range_Constraint;
```

Definition specifies the definition to query.

Returns the real_range_specification range_constraint of the definition.

Returns a Nil_Element if there is no explicit range_constraint.

Definition expects an element that has one of the following Definition_Kinds:

A_Constraint that has the following Constraint_Kinds:
  A_Digits_Constraint
  A_Delta_Constraint
A_Type_Definition that has one of the following Type_Kinds:

A_Floating_Point_Definition
An_Ordinary_Fixed_Point_Definition
A_Decimal_Fixed_Point_Definition

Raises ASIS_Inappropriate_Element with a Status of Value_Error for any element that does not have one of these expected kinds.

Returns an element that has one of the following Constraint_Kinds:

Not_A_Constraint
A_Simple_Expression_Range

16.14 function Index_Subtype_Definitions

```
function Index_Subtype_Definitions
            (Type_Definition : in Asis.Type_Definition)
            return Asis.Name_List;
```

Type_Definition specifies the array_type_definition to query.

Returns a list of the index_subtype_definition subtype mark names for an unconstrained_array_definition, in their order of appearance.

Type_Definition expects an element that has the following Type_Kinds:

An_Unconstrained_Array_Definition

or Type_Definition expects an element that has the following Formal_Type_Kinds:

A_Formal_Unconstrained_Array_Definition

Raises ASIS_Inappropriate_Element with a Status of Value_Error for any element that does not have one of these expected kinds.

Returns a list of elements that each have one of the following Expression_Kinds:

An_Identifier
A_Selected_Component

16.15 function Discrete_Subtype_Definitions

```
function Discrete_Subtype_Definitions
                    (Type_Definition : in Asis.Type_Definition)
                          return Asis.Definition_List;
```

Type_Definition specifies the array_type_definition to query.

Returns the list of Discrete_Subtype_Definition elements of a constrained_array_definition, in their order of appearance.

Type_Definition expects an element that has the following Type_Kinds:

A_Constrained_Array_Definition

or Type_Definition expects an element that has the following Formal_Type_Kinds:

A_Formal_Constrained_Array_Definition

Raises ASIS_Inappropriate_Element with a Status of Value_Error for any element that does not have one of these expected kinds.

Returns a list of elements that each have the following Definition_Kinds:

A_Discrete_Subtype_Definition

16.16 function Array_Component_Definition

```
function Array_Component_Definition
            (Type_Definition : in Asis.Type_Definition)
            return Asis.Component_Definition;
```

Type_Definition specifies the array_type_definition to query.

Returns the Component_Definition of the array_type_definition.

Type_Definition expects an element that has one of the following Type_Kinds:

An_Unconstrained_Array_Definition
A_Constrained_Array_Definition

or Type_Definition expects an element that has one of the following Formal_Type_Kinds:

A_Formal_Unconstrained_Array_Definition
A_Formal_Constrained_Array_Definition

Raises ASIS_Inappropriate_Element with a Status of Value_Error for any element that does not have one of these expected kinds.

Returns an element that has the following Definition_Kinds:

A_Component_Definition

16.17 function Access_To_Object_Definition

```
function Access_To_Object_Definition
               (Type_Definition : in Asis.Type_Definition)
                     return Asis.Subtype_Indication;
```

Type_Definition specifies the Access_Type_Definition to query.

Returns the subtype_indication following the reserved word **access**.

Type_Definition expects an element that has the following Type_Kinds:

An_Access_Type_Definition

or Type_Definition expects an element that has the following Formal_Type_Kinds:

A_Formal_Access_Type_Definition

or Type_Definition expects an element that has one of the following Access_Type_Kinds:

A_Pool_Specific_Access_To_Variable
An_Access_To_Variable
An_Access_To_Constant

Raises ASIS_Inappropriate_Element with a Status of Value_Error for any element that does not have one of these expected kinds.

Returns an element that has the following Element_Kinds:

A_Subtype_Indication

16.18 function Anonymous_Access_To_Object_Subtype_Mark

```
function Anonymous_Access_To_Object_Subtype_Mark
               (Definition : in Asis.Definition)
                     return Asis.Name;
```

Definition specifies the anonymous access definition to query.

Returns the subtype_mark following the reserved word(s) **access** or **access constant**.

Definition expects an element that has one of the following Definition_Kinds:

A_Pool_Specific_Access_To_Variable
An_Access_To_Variable
An_Access_To_Constant
An_Access_Definition that has one of the following Access_Definition_Kinds:
  An_Anonymous_Access_To_Variable
  An_Anonymous_Access_To_Constant

Raises ASIS_Inappropriate_Element with a Status of Value_Error for any element that does not have one of these expected kinds.

Returns an element that has one of the following Expression_Kinds:

An_Identifier
A_Selected_Component
An_Attribute_Reference

16.19 function Access_To_Subprogram_Parameter_Profile

```
function Access_To_Subprogram_Parameter_Profile
               (Type_Definition : in Asis.Type_Definition)
                     return Asis.Parameter_Specification_List;
```

Type_Definition specifies the Access_Type_Definition to query.

Returns a list of parameter_specification elements in the formal part of the parameter_profile in the access_to_subprogram_definition.

Returns a Nil_Element_List if the parameter_profile has no formal part.

Results of this query may vary across ASIS implementations. Some implementations normalize all multiple name parameter_specification elements into an equivalent sequence of corresponding single name parameter_specification elements. See Ada Standard 3.3.1(7).

Type_Definition expects an element that has either:

one of the following Type_Kinds:

An_Access_Type_Definition
A_Formal_Access_Type_Definition

or that has one of the following Access_Definition_Kinds:

An_Anonymous_Access_To_Procedure
An_Anonymous_Access_To_Protected_Procedure
An_Anonymous_Access_To_Function
An_Anonymous_Access_To_Protected_Function

Also if the kind is An_Access_Type_Definition or A_Formal_Access_Type_Definition then it also has one of the following Access_Type_Kinds:

An_Access_To_Procedure
An_Access_To_Protected_Procedure
An_Access_To_Function
An_Access_To_Protected_Function

Raises ASIS_Inappropriate_Element with a Status of Value_Error for any element that does not have one of these expected kinds.

Returns a list of elements that each have one of the following Declaration_Kinds:

A_Parameter_Specification

16.20 function Access_To_Function_Result_Subtype

```
function Access_To_Function_Result_Subtype
      (Definition : in Asis.Definition)
       return Asis.Definition;
```

Definition specifies the Access_Type_Definition or Access_Definition to query.

Returns a definition that corresponds to the result subtype of the access-to-function type, as specified by a subtype_indication (with no specified constraint) or an access_definition.

Definition expects an element that has either:

one of the following Type_Kinds:

An_Access_Type_Definition
A_Formal_Access_Type_Definition

or that has one of the following Access_Definition_Kinds:

An_Anonymous_Access_To_Function
An_Anonymous_Access_To_Protected_Function

Also if the kind is An_Access_Type_Definition or A_Formal_Access_Type_Definition then it also has one of the following Access_Type_Kinds:

An_Access_To_Function
An_Access_To_Protected_Function

Raises ASIS_Inappropriate_Element with a Status of Value_Error for any element that does not have one of these expected kinds.

Returns an element that has one of the following Definition_Kinds:

A_Subtype_Indication
An_Access_Definition

16.21 function Subtype_Mark

```
function Subtype_Mark (Definition : in Asis.Definition)
                       return Asis.Name;
```

Definition specifies the definition to query.

Returns the subtype_mark expression of the definition.

Definition expects an element that has one of the following Definition_Kinds:

A_Subtype_Indication
A_Discrete_Subtype_Definition that has the following Discrete_Range_Kinds:
  A_Discrete_Subtype_Indication
A_Discrete_Range that has the following Discrete_Range_Kinds:
  A_Discrete_Subtype_Indication
A_Formal_Type_Definition that has the following Formal_Type_Kinds:
  A_Formal_Derived_Type_Definition

Raises ASIS_Inappropriate_Element with a Status of Value_Error for any element that does not have one of these expected kinds.

Returns an element that has one of the following Expression_Kinds:

An_Identifier
A_Selected_Component
An_Attribute_Reference

16.22 function Subtype_Constraint

```
function Subtype_Constraint (Definition : in Asis.Definition)
                             return Asis.Constraint;
```

Definition specifies the definition to query.

Returns the constraint of the subtype_indication.

Returns a Nil_Element if no explicit constraint is present.

Definition expects an element that has one of the following Definition_Kinds:

A_Subtype_Indication
A_Discrete_Subtype_Definition that has the following Discrete_Range_Kinds:
  A_Discrete_Subtype_Indication
A_Discrete_Range that has the following Discrete_Range_Kinds:
  A_Discrete_Subtype_Indication

Raises ASIS_Inappropriate_Element with a Status of Value_Error for any element that does not have one of these expected kinds.

Returns an element that has one of the following Definition_Kinds:

Not_A_Definition
A_Constraint

16.23 function Lower_Bound

```
function Lower_Bound (Constraint : in Asis.Range_Constraint)
                     return Asis.Expression;
```

Constraint specifies the range_constraint or discrete_range to query.

Returns the simple_expression for the lower bound of the range.

Constraint expects an element that has the following Constraint_Kinds:

A_Simple_Expression_Range

or Constraint expects an element that has the following Discrete_Range_Kinds:

A_Discrete_Simple_Expression_Range

Raises ASIS_Inappropriate_Element with a Status of Value_Error for any element that does not have one of these expected kinds.

Returns an element that has the following Element_Kinds:

An_Expression

16.24 function Upper_Bound

```
function Upper_Bound (Constraint : in Asis.Range_Constraint)
                     return Asis.Expression;
```

Constraint specifies the range_constraint or discrete_range to query.

Returns the simple_expression for the upper bound of the range.

Constraint expects an element that has the following Constraint_Kinds:

A_Simple_Expression_Range

or Constraint expects an element that has the following Discrete_Range_Kinds:

A_Discrete_Simple_Expression_Range

Raises ASIS_Inappropriate_Element with a Status of Value_Error for any element that does not have one of these expected kinds.

Returns an element that has the following Element_Kinds:

An_Expression

16.25 function Range_Attribute

```
function Range_Attribute (Constraint : in Asis.Range_Constraint)
                         return Asis.Expression;
```

Constraint specifies the range_attribute_reference or discrete_range attribute_reference to query.

Returns the range_attribute_reference expression of the range.

Constraint expects an element that has the following Constraint_Kinds:

A_Range_Attribute_Reference

or Constraint expects an element that has the following Discrete_Range_Kinds:

A_Discrete_Range_Attribute_Reference

Raises ASIS_Inappropriate_Element with a Status of Value_Error for any element that does not have one of these expected kinds.

Returns an element that has the following Expression_Kinds:

An_Attribute_Reference

16.26 function Discrete_Ranges

```
function Discrete_Ranges (Constraint : in Asis.Constraint)
                         return Asis.Discrete_Range_List;
```

Constraint specifies the array index_constraint to query.

Returns the list of discrete_range components for an index_constraint, in their order of appearance.

Constraint expects an element that has the following Constraint_Kinds:

An_Index_Constraint

Raises ASIS_Inappropriate_Element with a Status of Value_Error for any element that does not have one of these expected kinds.

Returns a list of elements that each have the following Definition_Kinds:

A_Discrete_Range

16.27 function Discriminant_Associations

```
function Discriminant_Associations
            (Constraint : in Asis.Constraint;
             Normalized : in Boolean := False)
           return Asis.Discriminant_Association_List;
```

Constraint specifies the discriminant_constraint to query. Normalized specifies whether the normalized form is desired

Returns a list of the discriminant_association elements of the discriminant_constraint.

Returns a Nil_Element_List if there are no discriminant_association elements.

An unnormalized list contains only explicit associations ordered as they appear in the program text. Each unnormalized association has a list of discriminant_selector_name elements and an explicit expression.

A normalized list contains artificial associations representing all explicit associations. It has a length equal to the number of discriminant_specification elements of the known_discriminant_part. The order of normalized associations matches the order of discriminant_specification elements.

Each normalized association represents a one-on-one mapping of a discriminant_specification to the explicit expression. A normalized association has one A_Defining_Name component that denotes the discriminant_specification, and one An_Expression component that is the explicit expression.

Constraint expects an element that has the following Constraint_Kinds:

A_Discriminant_Constraint

Raises ASIS_Inappropriate_Element with a Status of Value_Error for any element that does not have one of these expected kinds.

Returns a list of elements that each have the following Association_Kinds:

A_Discriminant_Association

Implementation Requirements

Normalized associations are Is_Normalized and Is_Part_Of_Implicit. Normalized associations are never Is_Equal to unnormalized associations.

16.28 function Component_Subtype_Indication

```
function Component_Subtype_Indication
        (Component_Definition : in Asis.Component_Definition)
            return Asis.Subtype_Indication;
```

Component_Definition specifies the Component_Definition to query.

Returns the subtype_indication of the Component_Definition.

Component_Definition expects an element that has the following Definition_Kinds:

A_Component_Definition

Raises ASIS_Inappropriate_Element with a Status of Value_Error for any element that does not have one of these expected kinds.

Returns an element that has the following Definition_Kinds:

A_Subtype_Indication

16.29 function Discriminants

```
function Discriminants (Definition : in Asis.Definition)
                       return Asis.Discriminant_Specification_List;
```

Definition specifies the known_discriminant_part to query.

Returns a list of discriminant_specification elements, in their order of appearance.

Results of this query may vary across ASIS implementations. Some implementations normalize all multi-name discriminant_specification elements into an equivalent sequence of single name discriminant_specification elements. See Ada Standard 3.3.1(7).

Definition expects an element that has the following Definition_Kinds:

A_Known_Discriminant_Part

Raises ASIS_Inappropriate_Element with a Status of Value_Error for any element that does not have one of these expected kinds.

Returns a list of elements that each have the following Declaration_Kinds:

A_Discriminant_Specification

16.30 function Record_Components (definition)

```
function Record_Components (Definition : in Asis.Definition;
                            Include_Pragmas : in Boolean := False)
                            return Asis.Record_Component_List;
```

Definition specifies the record_definition or variant to query. Include_Pragmas specifies whether pragmas are to be returned.

Returns a list of the components and pragmas of the record_definition or variant, in their order of appearance.

Declarations are not returned for implementation-defined components of the record_definition. See Ada Standard 13.5.1 (15). These components are not normally visible to the ASIS application. However, they can be obtained with the query Implicit_Components.

Definition expects an element that has one of the following Definition_Kinds:

A_Record_Definition
A_Variant

Raises ASIS_Inappropriate_Element with a Status of Value_Error for any element that does not have one of these expected kinds.

Returns a list of elements that each have one of the following Element_Kinds:

A_Pragma
A_Declaration that has the following Declaration_Kinds:
  A_Component_Declaration
A_Definition that has one of the following Definition_Kinds:
  A_Null_Component
  A_Variant_Part
A_Clause that has the following Definition_Kinds:
  An_Attribute_Definition_Clause

16.31 function Implicit_Components

```
function Implicit_Components
                (Definition : in Asis.Definition)
                    return Asis.Record_Component_List;
```

Definition specifies the record_definition or variant to query.

Returns a list of all implicit implementation-defined components of the record_definition or variant. The Enclosing_Element of each component is the Definition argument. Each component is Is_Part_Of_Implicit.

Returns a Nil_Element_List if there are no implicit implementation-defined components or if the ASIS implementation does not support such implicit declarations.

Definition expects an element that has one of the following Definition_Kinds:

A_Record_Definition

A_Variant

Raises ASIS_Inappropriate_Element with a Status of Value_Error for any element that does not have one of these expected kinds.

Returns a list of elements that each have the following Declaration_Kinds:

A_Component_Declaration

Implementation Permissions

Some implementations do not represent all forms of implicit declarations such that elements representing them can be easily provided. An implementation can choose whether or not to construct and provide artificial declarations for implicitly declared elements.

Use the query Implicit_Components_Supported to determine if the implementation provides implicit record components.

16.32 function Discriminant_Direct_Name

```
function Discriminant_Direct_Name
            (Variant_Part : in Asis.Record_Component)
            return Asis.Name;
```

Variant_Part specifies the variant_part to query.

Returns the Discriminant_Direct_Name of the variant_part.

Variant_Part expects an element that has the following Definition_Kinds:

A_Variant_Part

Raises ASIS_Inappropriate_Element with a Status of Value_Error for any element that does not have one of these expected kinds.

Returns an element that has the following Expression_Kinds:

An_Identifier

16.33 function Variants

```
function Variants (Variant_Part    : in Asis.Record_Component;
                   Include_Pragmas : in Boolean := False)
                   return Asis.Variant_List;
```

Variant_Part specifies the variant_part to query. Include_Pragmas specifies whether pragmas are to be returned.

Returns a list of variants that make up the record component, in their order of appearance.

The only pragmas returned are those following the reserved word **is** and preceding the reserved word **when** of first variant, and those between following variants.

Variant_Part expects an element that has the following Definition_Kinds:

A_Variant_Part

Raises ASIS_Inappropriate_Element with a Status of Value_Error for any element that does not have one of these expected kinds.

Returns a list of elements that each have one of the following Element_Kinds:

A_Pragma
A_Definition that has the following Definition_Kinds:
  A_Variant

16.34 function Variant_Choices

```
function Variant_Choices (Variant : in Asis.Variant)
                         return Asis.Element_List;
```

Variant specifies the variant to query.

Returns the discrete_choice_list elements, in their order of appearance. Choices are either an expression, a discrete range, or an others choice.

Variant expects an element that has the following Definition_Kinds:

A_Variant

Raises ASIS_Inappropriate_Element with a Status of Value_Error for any element that does not have one of these expected kinds.

Returns a list of elements that each have one of the following Element_Kinds:

An_Expression
A_Definition that has one of the following Definition_Kinds:
A_Discrete_Range
An_Others_Choice

16.35 function Ancestor_Subtype_Indication

```
function Ancestor_Subtype_Indication
               (Definition : in Asis.Definition)
                     return Asis.Subtype_Indication;
```

Definition specifies the definition to query.

Returns the ancestor_subtype_indication following the reserved word **new** in the private_extension_declaration.

Definition expects an element that has the following Definition_Kinds:

A_Private_Extension_Definition

Raises ASIS_Inappropriate_Element with a Status of Value_Error for any element that does not have one of these expected kinds.

Returns an element that has the following Definition_Kinds:

A_Subtype_Indication

16.36 function Visible_Part_Items

```
function Visible_Part_Items
            (Definition : in Asis.Definition;
             Include_Pragmas : in Boolean := False)
           return Asis.Declarative_Item_List;
```

Type_Definition specifies the type_definition to query. Include_Pragmas specifies whether pragmas are to be returned.

Returns a list of declarations, representation and operational items, and pragmas in the visible part of the task or protected definition, in their order of appearance. The list does not include discriminant_specification elements of the known_discriminant_part, if any, of the protected type or task type declaration.

Returns a Nil_Element_List if there are no items.

Definition expects an element that has one of the following Definition_Kinds:

A_Task_Definition

A_Protected_Definition

Raises ASIS_Inappropriate_Element with a Status of Value_Error for any element that does not have one of these expected kinds.

Returns a list of elements that each have one of the following Element_Kinds:

A_Pragma
A_Declaration
A_Clause

16.37 function Private_Part_Items

```
function Private_Part_Items
            (Definition : in Asis.Definition;
             Include_Pragmas : in Boolean := False)
             return Asis.Declarative_Item_List;
```

Type_Definition specifies the task or protected definition to query. Include_Pragmas specifies whether pragmas are to be returned.

Returns a list of declarations, representation and operational items, and pragmas in the private part of the task or protected definition, in their order of appearance.

Returns a Nil_Element_List if there are no items.

Definition expects an element that has one of the following Definition_Kinds:

A_Task_Definition
A_Protected_Definition

Raises ASIS_Inappropriate_Element with a Status of Value_Error for any element that does not have one of these expected kinds.

Returns a list of elements that each have one of the following Element_Kinds:

A_Pragma
A_Declaration
A_Clause

16.38 function Is_Private_Present (definition)

```
function Is_Private_Present
            (Definition : in Asis.Definition) return Boolean;
```

Definition specifies the definition to query.

Returns True if the argument is a task_definition or a protected_definition that has a reserved word **private** marking the beginning of a (possibly empty) private part.

Returns False for any definition without a private part. Returns False for any unexpected Element.

Definition expects an element that has one of the following Definition_Kinds:

A_Task_Definition
A_Protected_Definition

Raises ASIS_Inappropriate_Element with a Status of Value_Error for any element that does not have one of these expected kinds.

16.39 function Is_Task_Definition_Present

```
function Is_Task_Definition_Present
      (Definition : in Asis.Definition)
       return Boolean;
```

Definition specifies the definition element to query.

Returns True if the element has a task_definition that is given explicitly.

Returns False for any other Element including a Nil_Element.

Definition expects an element that has the following Definition_Kinds:

A_Task_Definition

NOTE Is_Task_Definition_Present is used to determine whether the original text was `task T;` (for which it returns False) or `task T is end T;` (for which it returns True).

16.40 function Progenitor_List (definition)

```
function Progenitor_List
      (Type_Definition : in Asis.Definition)
       return Asis.Name_List;
```

Type_Definition specifies the definition to query.

Returns a list of subtype marks making up the interface_list in the argument definition, in their order of appearance.

Type_Definition expects an element that has one of the following Type_Kinds:

A_Derived_Record_Extension_Definition
An_Interface_Type_Definition

or Type_Definition expects an element that has one of the following Formal_Type_Kinds:

A_Formal_Derived_Type_Definition
A_Formal_Interface_Type_Definition

Raises ASIS_Inappropriate_Element with a Status of Value_Error for any element that does not have one of these expected kinds.

Returns a list of elements that each has one of the following Expression_Kinds:

An_Identifier
A_Selected_Component

`

Section 17: package Asis.Expressions

The library package Asis.Expressions shall exist. The package shall provide interfaces equivalent to those described in the following subclauses.

Asis.Expressions encapsulates a set of queries that operate on An_Expression and An_Association elements.

17.1 function Corresponding_Expression_Type

```
function Corresponding_Expression_Type (Expression : in Asis.Expression)
                        return Asis.Declaration;
```

Expression specifies the expression to query.

Returns the declaration for the subtype named by the subtype mark in the definition of the nominal subtype of the expression. If the subtype mark in the definition is a Base attribute reference, the declaration of the prefix of the attribute is returned.

Returns an implementation-defined result if the expression is an attribute reference other than Base, aggregate, string literal, allocator, membership test, short-circuit operation, or the invocation of a predefined operator.

For a slice, the result is the same as for the prefix of the slice. For an invocation of a user-defined operator, the result is the same as for the equivalent function call. For a parenthesized expression, it is the same as for the enclosed expression.

Returns a Nil_Element if the expression is of an anonymous or classwide type, or is a named number, a numeric literal, or a null literal.

Returns a Nil_Element if the expression denotes an entity that does not have a type, such as a package or an exception.

Expression expects an element that has the following Element_Kinds:

An_Expression

Raises ASIS_Inappropriate_Element with a Status of Value_Error for any element that does not have one of these expected kinds.

Returns an element that has one of the following Element_Kinds:

Not_An_Element
A_Declaration

NOTE This query does not "unwind" subtypes or derived types to get to the first subtype or parent subtype declarations, but it does ignore explicit constraints or null exclusions in the definition of the nominal subtype. For example, for the following program text:

```
type Int is range -5_000 .. 5_000;
type My_Int is new Int;
type Good_Int is new My_Int;
Var: Good_Int range -2_000 .. 2_000;
```

For Corresponding_Expression_Type of an expression denoting Var, the declaration for Good_Int should be returned. No further "unwinding" should occur. The declaration for either My_Int or Int should not be returned.

17.2 function Value_Image

```
function Value_Image (Expression : in Asis.Expression) return Wide_String;
```

Expression specifies the expression to query.

Returns the string image of the value of the string, integer, or real literal.

For string literals, Value_Image returns the quotes around the string literal, these quotes are doubled, just as any quote appearing embedded in the string literal in the program text.

The form of numbers returned by this query may vary between implementors. Implementors are encouraged, but not required, to return numeric literals using the same based or exponent form used in the original compilation text.

Expression expects an element that has one of the following Expression_Kinds:

An_Integer_Literal
A_Real_Literal
A_String_Literal

Raises ASIS_Inappropriate_Element with a Status of Value_Error for any element that does not have one of these expected kinds.

17.3 function Name_Image

```
function Name_Image (Expression : in Asis.Expression) return Program_Text;
```

Expression specifies the name to query.

Returns the program text image of the name.

An_Operator_Symbol elements have names with embedded quotes ""abs"" (function "**abs**").

A_Character_Literal elements have names with embedded apostrophes "'x'" (literal 'x').

An_Enumeration_Literal and An_Identifier elements have identifier names "Blue" (literal Blue) "Abc" (identifier Abc).

The case of names returned by this query may vary between implementors. Implementors are encouraged, but not required, to return names in the same case as was used in the original compilation text.

Expression expects an element that has one of the following Expression_Kinds:

An_Identifier
An_Operator_Symbol
A_Character_Literal
An_Enumeration_Literal

Raises ASIS_Inappropriate_Element with a Status of Value_Error for any element that does not have one of these expected kinds.

NOTE Implicit subtypes that can be encountered while traversing the semantic information embedded in implicit inherited subprogram declarations (Ada Standard 3.4 (17-22)) could have names that are unique in a particular scope. This is because these subtypes are Is_Part_Of_Implicit declarations that do not form part of the physical text of the original compilation units. Some applications may wish to carefully separate the names of declarations from the names of Is_Part_Of_Implicit declaration when creating symbol tables and other name-specific lookup mechanisms.

17.4 function References

```
function References (Name           : in Asis.Element;
                     Within_Element : in Asis.Element;
                     Implicitly     : in Boolean := False)
                    return Asis.Name_List;
```

Name specifies the entity to query. Within_Element specifies the limits for the query which is limited to the Element and its children.

If the Implicitly argument is True:

Returns all usage references of the given entity made by both explicit and implicit elements within the given limits.

If the Implicitly argument is False:

Returns all usage references of the given entity made only by explicit elements within the given limits.

Returned references are in their order of appearance.

Name expects an element that has the following Element_Kinds:

A_Defining_Name

Raises ASIS_Inappropriate_Element with a Status of Value_Error for any element that does not have one of these expected kinds.

Within_Element expects any kind of element.

Returns a list of elements that each have the following Element_Kinds:

An_Expression

May raise ASIS_Failed with a Status of Obsolete_Reference_Error if the argument is part of an inconsistent compilation unit.

17.5 function Is_Referenced

```
function Is_Referenced (Name           : in Asis.Element;
                        Within_Element : in Asis.Element;
                        Implicitly     : in Boolean := False)
                       return Boolean;
```

Name specifies the entity to query. Within_Element specifies the limits for the query which is limited to the Element and its children.

If the Implicitly argument is True:

Returns True if the Name is referenced by either implicit or explicit elements within the given limits, and returns False otherwise.

If the Implicitly argument is False:

Returns True only if the Name is referenced by explicit elements, and returns False otherwise.

Returns False for any unexpected Element.

Name expects an element that has the following Element_Kinds:

A_Defining_Name

Within_Element expects any kind of element.

May raise ASIS_Failed with a Status of Obsolete_Reference_Error if the argument is part of an inconsistent compilation unit.

17.6 function Corresponding_Name_Definition

```
function Corresponding_Name_Definition (Reference : in Asis.Expression)
                         return Asis.Defining_Name;
```

Reference specifies an expression to query.

Returns the defining_identifier, defining_character_literal, defining_operator_symbol, or defining_program_unit_name from the declaration of the referenced entity.

- Record component references return the defining name of the record discriminant or component_declaration. For references to inherited declarations of derived

types, the Corresponding_Name_Definition returns the defining name of the implicit inherited declaration.

- References to implicit operators and inherited subprograms will return an Is_Part_Of_Implicit defining name for the operation. The Enclosing_Element of the name is an implicit declaration for the operation. The Enclosing_Element of the declaration is the associated derived_type_definition.
- References to formal parameters given in calls to inherited subprograms will return an Is_Part_Of_Implicit defining name for the Parameter_Specification from the inherited subprogram specification.
- References to visible components of instantiated generic packages will return a name from the expanded generic specification instance.
- References, within expanded generic instances, that refer to other components of the same, or an enclosing, expanded generic instance, return a name from the appropriate expanded specification or body instance.

In case of renaming, the function returns the new name for the entity.

Returns a Nil_Element if the reference is to an implicitly declared element for which the implementation does not provide declarations and defining name elements.

Returns a Nil_Element if the argument is a dispatching call.

The Enclosing_Element of a non-Nil result is either a Declaration or a Statement.

Reference expects an element that has one of the following Expression_Kinds:

An_Identifier
An_Operator_Symbol
A_Character_Literal
An_Enumeration_Literal

Raises ASIS_Inappropriate_Element with a Status of Value_Error for any element that does not have one of these expected kinds.

Returns an element that has one of the following Element_Kinds:

Not_An_Element
A_Defining_Name

Implementation Permissions

An implementation may choose to return any part of multi-part declarations and definitions. Multi-part declaration/definitions can occur for:

- Subprogram specification in package specification, package body, and subunits (**is separate**);
- Entries in package specification, package body, and subunits (**is separate**);
- Private type and full type declarations;
- Incomplete type and full type declarations; and
- Deferred constant and full constant declarations.

No guarantee is made that the element will be the first part or that the determination will be made due to any visibility rules. An application should make its own analysis for each case based on which part is returned.

Some implementations do not represent all forms of implicit declarations such that elements representing them can be easily provided. An implementation can choose whether or not to construct and provide artificial declarations for implicitly declared elements.

Implementation Requirements

Raises ASIS_Inappropriate_Element, with a Status of Value_Error, if passed a reference that does not have a declaration, including the following:

- a reference to an attribute_designator. Attributes are defined, but have no implicit or explicit declarations;
- an identifier which syntactically is placed before "=>" in a pragma_argument_association which has the form of a named association; such an identifier can never have a declaration;
- an identifier specific to a pragma (Ada Standard, 2.8(10)).

For example:

```
pragma Should_I_Check (Really => Yes);
```

In this example, both the names Really and Yes have no declaration.

Raises ASIS_Inappropriate_Element, with a Status of Value_Error, if passed a portion of a pragma that was "ignored" by the compiler and which does not have (sufficient) semantic information for a proper return result to be computed. For example:

```
pragma I_Am_Ignored (Foof);
```

The "Foof" expression is An_Identifier but raises this exception if passed to Corresponding_Name_Definition if the pragma was ignored or unprocessed.

Raises ASIS_Inappropriate_Element, with a Status of Value_Error, if passed a portion of a pragma that is an ambiguous reference to more than one entity. For example:

```
pragma Inline ("+");   -- Inlines all "+" operators
```

The "+" expression is An_Operator_Symbol but raises this exception if it referenced more than one "+" operator. In this case, the Corresponding_Name_Definition_List query can be used to obtain a list of referenced entities.

17.7 function Corresponding_Name_Definition_List

```
function Corresponding_Name_Definition_List (Reference : in Asis.Element)
                                 return Asis.Defining_Name_List;
```

Reference specifies an entity reference to query.

Exactly like Corresponding_Name_Definition except it returns a list. The list will almost always have a length of one. The exception to this is the case where an expression in a pragma is ambiguous and reference more than one entity. For example:

```
pragma Inline ("+");          -- Inlines all "+" operators
```

The "+" expression is An_Operator_Symbol but could reference more than one "+" operator. In this case, the resulting list includes all referenced entities.

Reference expects an element that has one of the following Expression_Kinds:

An_Identifier
An_Operator_Symbol
A_Character_Literal
An_Enumeration_Literal

Raises ASIS_Inappropriate_Element with a Status of Value_Error for any element that does not have one of these expected kinds.

Returns a list of elements that each have the following Element_Kinds:

A_Defining_Name

17.8 function Corresponding_Name_Declaration

```
function Corresponding_Name_Declaration (Reference : in Asis.Expression)
                          return Asis.Element;
```

Reference specifies the entity reference to query.

Returns the declaration that declared the entity named by the given reference. The result is exactly the same as:

```
Result := Corresponding_Name_Definition (Reference);
if not Is_Nil (Result) then
   Result := Enclosing_Element (Result);
end if;
return Result;
```

See the comments for Corresponding_Name_Definition for details. The result is either a Declaration or a Statement. Statements result from references to statement labels, loop identifiers, and block identifiers.

Reference expects an element that has one of the following Expression_Kinds:

An_Identifier
An_Operator_Symbol
A_Character_Literal
An_Enumeration_Literal

Raises ASIS_Inappropriate_Element with a Status of Value_Error for any element that does not have one of these expected kinds.

Returns an element that has one of the following Element_Kinds:

A_Declaration
A_Statement

Predefined types, exceptions, operators in package Standard can be checked by testing that the enclosing Compilation_Unit is standard.

17.9 function Prefix

```
function Prefix (Expression : in Asis.Expression) return Asis.Expression;
```

Expression specifies the name expression to query.

Returns the prefix (the construct to the left of: the rightmost unnested left parenthesis in function_call elements and indexed_component elements or slice elements, the rightmost 'dot' for selected_component elements, or the rightmost tick for attribute_reference elements).

Returns the operator_symbol for infix operator function calls. The infix form A + B is equivalent to the prefix form "+"(A, B).

Expression expects an element that has one of the following Expression_Kinds:

An_Explicit_Dereference — P.ALL
An_Implicit_Dereference — P.X, P'Attr, P(...)
An_Attribute_Reference — Priv'Base'First
A_Function_Call — Abc(...) or Integer'Image(...)
An_Indexed_Component — An_Array(3)
A_Selected_Component — A.B.C
A_Slice — An_Array(3 .. 5)

Raises ASIS_Inappropriate_Element with a Status of Value_Error for any element that does not have one of these expected kinds.

Returns an element that has the following Expression_Kinds:

An_Expression

17.10 function Index_Expressions

```
function Index_Expressions (Expression : in Asis.Expression)
                           return Asis.Expression_List;
```

Expression specifies an indexed_component to query.

Returns the list of expressions (possibly only one) within the parenthesis, in their order of appearance.

Expression expects an element that has the following Expression_Kinds:

An_Indexed_Component

Raises ASIS_Inappropriate_Element with a Status of Value_Error for any element that does not have one of these expected kinds.

Returns a list of elements that each have the following Element_Kinds:

An_Expression

17.11 function Slice_Range

```
function Slice_Range (Expression : in Asis.Expression)
                     return Asis.Discrete_Range;
```

Expression specifies the slice to query.

Returns the discrete range of the slice.

Expression expects an element that has the following Expression_Kinds:

A_Slice

Raises ASIS_Inappropriate_Element with a Status of Value_Error for any element that does not have one of these expected kinds.

Returns an element that has the following Definition_Kinds:

A_Discrete_Range

17.12 function Selector

```
function Selector (Expression : in Asis.Expression)
                  return Asis.Expression;
```

Expression specifies the selected_component to query.

Returns the selector (the construct to the right of the rightmost 'dot' in the selected_component).

Expression expects an element that has the following Expression_Kinds:

A_Selected_Component

Raises ASIS_Inappropriate_Element with a Status of Value_Error for any element that does not have one of these expected kinds.

Returns an element that has one of the following Expression_Kinds:

An_Identifier
An_Operator_Symbol
A_Character_Literal

An_Enumeration_Literal

17.13 function Attribute_Designator_Identifier

```
function Attribute_Designator_Identifier
        (Expression : in Asis.Expression)
            return Asis.Expression;
```

Expression specifies an attribute_reference expression to query.

Returns the identifier of the attribute_designator (the construct to the right of the rightmost tick of the attribute_reference). The Prefix of the attribute_reference can itself be an attribute_reference as in T'Base'First where the prefix is T'Base and the attribute_designator name is First.

Attribute_designator reserved words **access**, **delta**, **digits**, and **mod** are treated as An_Identifier.

Expression expects an element that has the following Expression_Kinds:

An_Attribute_Reference

Raises ASIS_Inappropriate_Element with a Status of Value_Error for any element that does not have one of these expected kinds.

Returns an element that has the following Expression_Kinds:

An_Identifier

17.14 function Attribute_Designator_Expressions

```
function Attribute_Designator_Expressions
        (Expression : in Asis.Expression)
            return Asis.Expression_List;
```

Expression specifies an attribute expression to query.

Returns the static expressions associated with the optional argument of the attribute_designator. Expected predefined attributes are A'First(N), A'Last(N), A'Length(N), and A'Range(N).

Returns a Nil_Element_List if there are no arguments.

Expression expects an element that has the following Expression_Kinds:

An_Attribute_Reference that has one of the following Attribute_Kinds:
A_First_Attribute
A_Last_Attribute
A_Length_Attribute
A_Range_Attribute
An_Implementation_Defined_Attribute
An_Unknown_Attribute

Raises ASIS_Inappropriate_Element with a Status of Value_Error for any element that does not have one of these expected kinds.

Returns a list of elements that each have the following Element_Kinds:

An_Expression

Implementation Permissions

This query returns a list to support implementation-defined attributes that may have more than one static_expression.

17.15 function Record_Component_Associations

```
function Record_Component_Associations
              (Expression : in Asis.Expression;
               Normalized : in Boolean := False)
                    return Asis.Association_List;
```

Expression specifies an aggregate expression to query. Normalized specifies whether the normalized form is desired.

Returns a list of the record_component_association elements of a record_aggregate or an extension_aggregate.

Returns a Nil_Element_List if the aggregate is of the form (null record).

An unnormalized list contains all needed associations ordered as they appear in the program text. Each unnormalized association has an optional list of discriminant_selector_name elements, and an explicit expression.

A normalized list contains artificial associations representing all needed components in an order matching the declaration order of the needed components.

Each normalized association represents a one on one mapping of a component to the explicit expression. A normalized association has one A_Defining_Name component that denotes the discriminant_specification or component_declaration, and one An_Expression component that is the expression.

Expression expects an element that has one of the following Expression_Kinds:

A_Record_Aggregate
An_Extension_Aggregate

Raises ASIS_Inappropriate_Element with a Status of Value_Error for any element that does not have one of these expected kinds.

Returns a list of elements that each have the following Association_Kinds:

A_Record_Component_Association

Implementation Requirements

Normalized associations are Is_Normalized and Is_Part_Of_Implicit. Normalized associations are never Is_Equal to unnormalized associations.

17.16 function Extension_Aggregate_Expression

```
function Extension_Aggregate_Expression
                         (Expression : in Asis.Expression)
                                  return Asis.Expression;
```

Expression specifies an extension_aggregate expression to query.

Returns the ancestor_part expression preceding the reserved word **with** in the extension_aggregate.

Expression expects an element that has the following Expression_Kinds:

An_Extension_Aggregate

Raises ASIS_Inappropriate_Element with a Status of Value_Error for any element that does not have one of these expected kinds.

Returns an element that has the following Element_Kinds:

An_Expression

17.17 function Array_Component_Associations

```
function Array_Component_Associations
              (Expression : in Asis.Expression)
                    return Asis.Association_List;
```

Expression specifies an array aggregate expression to query.

Returns a list of the Array_Component_Associations in an array aggregate. If the aggregate is a positional array aggregate, the Array_Component_Associations consist of an expression of the aggregate with Array_Component_Choices that are each a Nil_Element_List for all positional expressions except for the others choice, if any.

Expression expects an element that has one of the following Expression_Kinds:

A_Positional_Array_Aggregate
A_Named_Array_Aggregate

Raises ASIS_Inappropriate_Element with a Status of Value_Error for any element that does not have one of these expected kinds.

Returns a list of elements that each have the following Association_Kinds:

An_Array_Component_Association

NOTE While positional_array_aggregate elements do not have array_component_association elements defined by Ada syntax, ASIS treats A_Positional_Array_Aggregate as if it were A_Named_Array_Aggregate.

17.18 function Array_Component_Choices

```
function Array_Component_Choices
        (Association : in Asis.Association)
            return Asis.Expression_List;
```

Association specifies the component association to query.

If the Association is from a named_array_aggregate:

- Returns the discrete_choice_list in order of appearance in the program text. The choices are either An_Expression or A_Discrete_Range elements, or a single An_Others_Choice element.

If the Association is from a positional_array_aggregate:

- Returns a single An_Others_Choice if the association is an others choice (**others** => expression).
- Returns a Nil_Element_List otherwise.

Association expects an element that has the following Association_Kinds:

An_Array_Component_Association

Raises ASIS_Inappropriate_Element with a Status of Value_Error for any element that does not have one of these expected kinds.

Returns a list of elements that each have one of the following Element_Kinds:

An_Expression
A_Definition that has one of the following Definition_Kinds:
A_Discrete_Range
An_Others_Choice

17.19 function Record_Component_Choices

```
function Record_Component_Choices
        (Association : in Asis.Association)
            return Asis.Expression_List;
```

Association specifies the component association to query.

If the Association argument is from an unnormalized list:

- If the Association is a named component association:
  - Returns the component_choice_list in order of appearance in the program text. The choices are either An_Identifier elements representing component_selector_name elements, or a single An_Others_Choice element.
  - The Enclosing_Element of the choices is the Association argument.
- If the Association is a positional component association:
  - Returns a Nil_Element_List.

If the Association argument is from a normalized list:

- Returns a list containing a single choice:
  - A_Defining_Name element representing the defining_identifier of the component_declaration.
  - The Enclosing_Element of the A_Defining_Name is the component_declaration.

Normalized lists contain artificial ASIS An_Association elements that provide one formal A_Defining_Name => An_Expression pair per association. These artificial associations are Is_Normalized. Their component A_Defining_Name is not Is_Normalized.

Association expects an element that has the following Association_Kinds:

A_Record_Component_Association

Raises ASIS_Inappropriate_Element with a Status of Value_Error for any element that does not have one of these expected kinds.

Returns a list of elements that each have one of the following Element_Kinds:

A_Defining_Name — Is_Normalized(Association)
An_Expression — **not** Is_Normalized(Association)
 that has the following Expression_Kinds:
  An_Identifier
A_Definition
 that has the following Definition_Kinds:
  An_Others_Choice

17.20 function Component_Expression

```
function Component_Expression (Association : in Asis.Association)
                               return Asis.Expression;
```

Association specifies the component association to query.

Returns the expression of the record_component_association or array_component_association.

If the Association argument is from a normalized list, the Enclosing_Element of the returned expression is the unnormalized An_Association Element containing the corresponding component association. Otherwise, the Enclosing_Element of the returned expression is the Association argument

Normalized lists contain artificial ASIS An_Association elements that provide one formal A_Defining_Name => An_Expression pair per association. These artificial associations are Is_Normalized. Their component An_Expression elements are not Is_Normalized.

For An_Array_Component_Association and unnormalized A_Record_Component_Association where the association contains a box expression, Asis.Expressions.Component_Expression returns A_Box_Expression.

For a normalized A_Record_Component_Association, where the association contains a box expression, if the corresponding record type that contains this component contains a default expression, Asis.Expressions.Component_Expression returns this default expression, otherwise Asis.Expressions.Component_Expression returns A_Box_Expression.

Association expects an element that has one of the following Association_Kinds:

A_Record_Component_Association
An_Array_Component_Association

Raises ASIS_Inappropriate_Element with a Status of Value_Error for any element that does not have one of these expected kinds.

Returns an element that has the following Element_Kinds:

An_Expression

17.21 function Formal_Parameter

```
function Formal_Parameter (Association : in Asis.Association)
                           return Asis.Element;
```

Association specifies the association to query.

If the Association argument is from an unnormalized list:

- If the Association is given in named notation:

  Returns An_Identifier representing the formal_parameter_selector_name, generic_formal_parameter_selector_name, or pragma_argument_identifier.

  The Enclosing_Element of the An_Identifier element is the Association argument.

- If the Association is given in positional notation:

  Returns a Nil_Element.

If the Association argument is from a normalized list:

- Returns A_Defining_Name representing the defining_identifier of the parameter_specification or generic_formal_parameter_declaration. Pragma_argument_association elements are not available in normalized form.
- The Enclosing_Element of the A_Defining_Name is the parameter_specification or generic_formal_parameter_declaration element.

Normalized lists contain artificial ASIS An_Association elements that provide one formal A_Defining_Name => An_Expression pair per association. These artificial associations are Is_Normalized. Their component A_Defining_Name elements are not Is_Normalized.

Asis.Expressions.Formal_Parameter may return An_Others_Choice for a unnormalized A_Generic_Association argument;

If a formal_package_association contains a box, then the corresponding unnormalized A_Generic_Association element contains an Expression_Element with expression kind A_Box_Expression as its Actual_Parameter part. The normalized A_Generic_Association

contains either a default parameter or an Expression_Element with expression kind A_Box_Expression;

Association expects an element that has one of the following Association_Kinds:

A_Parameter_Association
A_Generic_Association
A_Pragma_Argument_Association

Raises ASIS_Inappropriate_Element with a Status of Value_Error for any element that does not have one of these expected kinds.

Returns an element that has one of the following Element_Kinds:

Not_An_Element
An_Operator_Symbol
A_Defining_Name — Is_Normalized(Association)
An_Expression — **not** Is_Normalized(Association)

Returns an element that has the following Expression_Kinds:

An_Identifier

17.22 function Actual_Parameter

```
function Actual_Parameter (Association : in Asis.Association)
                          return Asis.Expression;
```

Association specifies the association to query

If the Association argument is from an unnormalized list:

- Returns An_Expression representing:
  - the explicit_actual_parameter of a parameter_association.
  - the explicit_generic_actual_parameter of a generic_association.
  - the name or expression of a pragma_argument_association.

  The Enclosing_Element of An_Expression is the Association argument.

If the Association argument is from a normalized list:

- If the Association is given explicitly:
  - Returns An_Expression representing:
    - the explicit_actual_parameter of a parameter_association.
    - the explicit_generic_actual_parameter of a generic_association.

    The Enclosing_Element of An_Expression is the unnormalized An_Association Element containing the corresponding actual parameter
- If the Association is given by default:
  - Returns An_Expression representing:
    - the corresponding default_expression of the Is_Normalized A_Parameter_Association.
    - the corresponding default_expression or default_name of the Is_Normalized A_Generic_Association.
  - The Enclosing_Element of the An_Expression element is the parameter_specification or generic_formal_parameter_declaration that contains the default_expression or default_name, except for the case when this An_Expression element is an implicit naming expression representing the actual subprogram selected at the place of the instantiation for

A_Box_Default. In the latter case, the Enclosing_Element for such An_Expression is the instantiation.

- Normalized lists contain artificial ASIS An_Association elements that provide one formal A_Defining_Name => An_Expression pair per association. These artificial associations are Is_Normalized. Artificial associations of default associations are Is_Defaulted_Association. Their component An_Expression elements are not Is_Normalized and are not Is_Defaulted_Association.

If the argument is A_Pragma_Argument_Association, then this function may return any expression to support implementation-defined pragmas.

Association expects an element that has one of the following Association_Kinds:

A_Parameter_Association
A_Generic_Association
A_Pragma_Argument_Association

Raises ASIS_Inappropriate_Element with a Status of Value_Error for any element that does not have one of these expected kinds.

Returns an element that has the following Element_Kinds:

An_Expression

17.23 function Discriminant_Selector_Names

```
function Discriminant_Selector_Names
            (Association : in Asis.Discriminant_Association)
           return Asis.Expression_List;
```

Association specifies the discriminant association to query.

If the Association argument is from an unnormalized list:

- If the Association is a named discriminant_association:
  - Returns a list of the An_Identifier discriminant_selector_name elements in order of appearance.
  - The Enclosing_Element of the names is the Association argument.
- If the Association is a positional discriminant_association:
  - Returns a Nil_Element_List.

If the Association argument is from a normalized list:

- Returns a list containing a single A_Defining_Name element representing the defining_identifier of the discriminant_specification.
- The Enclosing_Element of the A_Defining_Name is the discriminant_specification.
- Normalized lists contain artificial ASIS An_Association elements that provide one formal A_Defining_Name => An_Expression pair per association. These artificial associations are Is_Normalized. Their component A_Defining_Name elements are not Is_Normalized.

Association expects an element that has the following Association_Kinds:

A_Discriminant_Association

Raises ASIS_Inappropriate_Element with a Status of Value_Error for any element that does not have one of these expected kinds.

Returns a list of elements that each one of the following Element_Kinds:

A_Defining_Name — Is_Normalized (Association)

An_Expression — **not** Is_Normalized (Association)
that has the following Expression_Kinds:
An_Identifier

17.24 function Discriminant_Expression

```
function Discriminant_Expression
            (Association : in Asis.Discriminant_Association)
            return Asis.Expression;
```

Association specifies the discriminant_association to query.

If the Association argument is from an unnormalized list:

- Returns An_Expression representing the expression of the discriminant_association.

  The Enclosing_Element of the returned An_Expression is the Association argument.

If the Association argument is from a normalized list:

- If the Association is given explicitly:
  - Returns An_Expression representing the expression of the discriminant_association.

    The Enclosing_Element of the returned An_Expression is the unnormalized An_Association Element containing the corresponding discriminant_specification.
- If the Association is given by default:
  - Returns An_Expression representing the corresponding default_expression of the Is_Normalized A_Discriminant_Association.

    The Enclosing_Element of the the returned An_Expression element is the discriminant_specification that contains the default_expression.
- Normalized lists contain artificial ASIS An_Association elements that provide one formal A_Defining_Name => An_Expression pair per association. These artificial associations are Is_Normalized. Artificial associations of default associations are Is_Defaulted_Association. Their component An_Expression elements are not Is_Normalized and are not Is_Defaulted_Association.

Association expects an element that has the following Association_Kinds:

A_Discriminant_Association

Raises ASIS_Inappropriate_Element with a Status of Value_Error for any element that does not have one of these expected kinds.

Returns an element that has the following Element_Kinds:

An_Expression

17.25 function Is_Normalized

```
function Is_Normalized (Association : in Asis.Association) return Boolean;
```

Association specifies the association to query.

Returns True if the association is a normalized, artificially created association returned by the queries Discriminant_Associations, Generic_Actual_Part, Call_Statement_Parameters, Record_Component_Associations, or Function_Call_Parameters where Normalized => True.

Returns False for any unexpected Element.

Association expects an element that has one of the following Association_Kinds:

A_Discriminant_Association
A_Record_Component_Association
A_Parameter_Association
A_Generic_Association

17.26 function Is_Defaulted_Association

```
function Is_Defaulted_Association
    (Association : in Asis.Association) return Boolean;
```

Association specifies the association to query.

Returns True if the association is a normalized, artificially created association returned by the queries Discriminant_Associations, Generic_Actual_Part, Record_Component_Associations, Call_Statement_Parameters, or Function_Call_Parameters where Normalized => True and the association contains a default expression. A default expression is one that is implicitly supplied by the language semantics and that was not explicitly supplied (typed) by the user.

Returns False for any unexpected Element.

Association expects an element that has one of the following Association_Kinds:

A_Parameter_Association
A_Generic_Association

17.27 function Expression_Parenthesized

```
function Expression_Parenthesized (Expression : in Asis.Expression)
                                  return Asis.Expression;
```

Expression specifies the parenthesized expression to query.

Returns the expression within the parenthesis. This operation unwinds only one set of parenthesis at a time, so the result may itself be A_Parenthesized_Expression.

A_Parenthesized_Expression kind corresponds only to the (expression) alternative in the syntax notion of primary in Ada Standard 4.4. For example, an expression of a type_conversion is A_Parenthesized_Expression only if it is similar to the form subtype_mark((expression)) where it has at least one set of its own parenthesis.

Expression expects an element that has the following Expression_Kinds:

A_Parenthesized_Expression

Raises ASIS_Inappropriate_Element with a Status of Value_Error for any element that does not have one of these expected kinds.

Returns an element that has the following Element_Kinds:

An_Expression

17.28 function Is_Prefix_Call

```
function Is_Prefix_Call (Expression : in Asis.Expression) return Boolean;
```

Expression specifies the function call expression to query.

Returns True if the function call is in prefix form.

Returns False for any unexpected Element.

For example:

```
Foo (A, B);    -- Returns True
"<" (A, B);    -- Returns True
... A < B ... -- Returns False
```

Expression expects an element that has the following Expression_Kinds:

A_Function_Call

17.29 function Corresponding_Called_Function

```
function Corresponding_Called_Function
        (Expression : in Asis.Expression)
            return Asis.Declaration;
```

Expression specifies the function_call to query.

Returns the declaration of the called function.

Returns a Nil_Element if the:

- *function*_prefix or the operator of an infix call denotes a predefined operator for which the implementation does not provide an artificial function declaration,
- prefix of the call denotes an implicit or explicit dereference of an access to a function value, or
- Expression is a dispatching call,

If the *function*_prefix or *function*_name denotes an attribute_reference, and if the corresponding attribute is (re)defined by an attribute definition clause, an implementation is encouraged, but not required, to return the definition of the corresponding subprogram whose name is used after **use** in this attribute definition clause. If an implementation cannot return such a subprogram definition, a Nil_Element should be returned. For an attribute reference which is not (re)defined by an attribute definition clause, a Nil_Element should be returned.

Expression expects an element that has the following Expression_Kinds:

A_Function_Call

Raises ASIS_Inappropriate_Element with a Status of Value_Error for any element that does not have one of these expected kinds.

Returns an element that has one of the following Declaration_Kinds:

Not_A_Declaration
A_Function_Declaration
A_Function_Body_Declaration
A_Function_Body_Stub
A_Function_Renaming_Declaration
A_Function_Instantiation
A_Formal_Function_Declaration
A_Generic_Function_Declaration

Implementation Permissions

An implementation may choose to return any part of multi-part declarations and definitions. Multi-part declaration/definitions can occur for:

- Subprogram specification in package specification, package body, and subunits (**is separate**);
- Entries in package specification, package body, and subunits (**is separate**);
- Private type and full type declarations;
- Incomplete type and full type declarations; and

- Deferred constant and full constant declarations.

No guarantee is made that the element will be the first part or that the determination will be made due to any visibility rules. An application should make its own analysis for each case based on which part is returned.

An implementation can choose whether or not to construct and provide artificial implicit declarations for predefined operators.

17.30 function Function_Call_Parameters

```
function Function_Call_Parameters (Expression : in Asis.Expression;
                                   Normalized : in Boolean := False)
                                  return Asis.Association_List;
```

Expression specifies the function call expression to query. Normalized specifies whether the normalized form is desired.

Returns a list of parameter_association elements of the call.

Returns a Nil_Element_List if there are no parameter_association elements.

An unnormalized list contains only explicit associations ordered as they appear in the program text. Each unnormalized association has an optional formal_parameter_selector_name and an explicit_actual_parameter component.

A normalized list contains artificial associations representing all explicit and default associations. It has a length equal to the number of parameter_specification elements of the formal_part of the parameter_and_result_profile. The order of normalized associations matches the order of parameter_specification elements.

Each normalized association represents a one-to-one mapping of a parameter_specification element to the explicit or default expression. A normalized association has one A_Defining_Name component that denotes the parameter_specification, and one An_Expression component that is either the explicit_actual_parameter, a default_expression, or when the call uses a prefixed view of the function, the prefix of the call.

If the prefix of the call denotes an implicit or explicit dereference of an access to a function value, normalized associations are constructed on the basis of the formal_part of the parameter_and_result_profile from the corresponding access_to_subprogram definition.

Returns Nil_Element for normalized associations in the case where the called function can be determined only dynamically (dispatching calls). ASIS cannot produce any meaningful result in this case.

The exception ASIS_Inappropriate_Element is raised when the function call is an attribute reference and Is_Normalized is True.

Expression expects an element that has the following Expression_Kinds:

A_Function_Call

Raises ASIS_Inappropriate_Element with a Status of Value_Error for any element that does not have one of these expected kinds.

Returns a list of elements that each have the following Element_Kinds:

A_Parameter_Association

Normalized associations are Is_Normalized and Is_Part_Of_Implicit. Normalized associations provided by default are Is_Defaulted_Association. Normalized associations are never Is_Equal to unnormalized associations.

17.31 function Short_Circuit_Operation_Left_Expression

```
function Short_Circuit_Operation_Left_Expression
            (Expression : in Asis.Expression)
            return Asis.Expression;
```

Expression specifies the short circuit operation to query.

Returns the expression preceding the reserved words **and then** or **or else** in the short circuit expression.

Expression expects an element that has one of the following Expression_Kinds:

An_And_Then_Short_Circuit
An_Or_Else_Short_Circuit

Raises ASIS_Inappropriate_Element with a Status of Value_Error for any element that does not have one of these expected kinds.

Returns an element that has the the following Element_Kinds:

An_Expression

17.32 function Short_Circuit_Operation_Right_Expression

```
function Short_Circuit_Operation_Right_Expression
            (Expression : in Asis.Expression)
            return Asis.Expression;
```

Expression specifies the short circuit operation to query.

Returns the expression following the reserved words **or else** or **and then** in the short circuit expression.

Expression expects an element that has one of the following Expression_Kinds:

An_And_Then_Short_Circuit
An_Or_Else_Short_Circuit

Raises ASIS_Inappropriate_Element with a Status of Value_Error for any element that does not have one of these expected kinds.

Returns an element that has the the following Element_Kinds:

An_Expression

17.33 function Membership_Test_Expression

```
function Membership_Test_Expression (Expression : in Asis.Expression)
                                     return Asis.Expression;
```

Expression specifies the membership test operation to query.

Returns the expression on the left hand side of the membership test.

Expression expects an element that has one of the following Expression_Kinds:

An_In_Range_Membership_Test
A_Not_In_Range_Membership_Test
An_In_Type_Membership_Test

A_Not_In_Type_Membership_Test

Raises ASIS_Inappropriate_Element with a Status of Value_Error for any element that does not have one of these expected kinds.

Returns an element that has the following Element_Kinds:

An_Expression

17.34 function Membership_Test_Range

```
function Membership_Test_Range
            (Expression : in Asis.Expression)
            return Asis.Range_Constraint;
```

Expression specifies the membership test operation to query.

Returns the range following the reserved words **in** or **not in** from the membership test.

Expression expects an element that has one of the following Expression_Kinds:

An_In_Range_Membership_Test
A_Not_In_Range_Membership_Test

Raises ASIS_Inappropriate_Element with a Status of Value_Error for any element that does not have one of these expected kinds.

Returns an element that has one of the following Constraint_Kinds:

A_Range_Attribute_Reference
A_Simple_Expression_Range

17.35 function Membership_Test_Subtype_Mark

```
function Membership_Test_Subtype_Mark
            (Expression : in Asis.Expression)
            return Asis.Name;
```

Expression specifies the membership test operation to query.

Returns the subtype_mark expression following the reserved words **in** or **not in** from the membership test.

Expression expects an element that has one of the following Expression_Kinds:

An_In_Type_Membership_Test
A_Not_In_Type_Membership_Test

Raises ASIS_Inappropriate_Element with a Status of Value_Error for any element that does not have one of these expected kinds.

Returns an element that has one of the following Expression_Kinds:

An_Identifier
A_Selected_Component
An_Attribute_Reference

17.36 function Converted_Or_Qualified_Subtype_Mark

```
function Converted_Or_Qualified_Subtype_Mark
               (Expression : in Asis.Expression)
                   return Asis.Name;
```

Expression specifies the type conversion or qualified expression to query.

Returns the subtype_mark expression that converts or qualifies the expression.

Expression expects an element that has one of the following Expression_Kinds:

A_Type_Conversion
A_Qualified_Expression

Raises ASIS_Inappropriate_Element with a Status of Value_Error for any element that does not have one of these expected kinds.

Returns an element that has one of the following Expression_Kinds:

An_Identifier
A_Selected_Component
An_Attribute_Reference

17.37 function Converted_Or_Qualified_Expression

```
function Converted_Or_Qualified_Expression
               (Expression : in Asis.Expression)
                   return Asis.Expression;
```

Expression specifies the type conversion or qualified expression to query.

Returns the expression being converted or qualified.

Expression expects an element that has one of the following Expression_Kinds:

A_Type_Conversion
A_Qualified_Expression

Raises ASIS_Inappropriate_Element with a Status of Value_Error for any element that does not have one of these expected kinds.

Returns an element that has the following Element_Kinds:

An_Expression

17.38 function Allocator_Subtype_Indication

```
function Allocator_Subtype_Indication (Expression : in Asis.Expression)
                                       return Asis.Subtype_Indication;
```

Expression specifies the allocator expression to query.

Returns the subtype indication for the object being allocated.

Expression expects an element that has the following Expression_Kinds:

An_Allocation_From_Subtype

Raises ASIS_Inappropriate_Element with a Status of Value_Error for any element that does not have one of these expected kinds.

Returns an element that has the following Definition_Kinds:

A_Subtype_Indication

17.39 function Allocator_Qualified_Expression

```
function Allocator_Qualified_Expression (Expression : in Asis.Expression)
                                         return Asis.Expression;
```

Expression specifies the allocator expression to query.

Returns the qualified expression for the object being allocated.

Expression expects an element that has the following Expression_Kinds:

An_Allocation_From_Qualified_Expression

Raises ASIS_Inappropriate_Element with a Status of Value_Error for any element that does not have one of these expected kinds.

Returns an element that has the following Expression_Kinds:

A_Qualified_Expression

Section 18: package Asis.Statements

The library package Asis.Statements shall exist. The package shall provide interfaces equivalent to those described in the following subclauses.

Asis.Statements encapsulates a set of queries that operate on A_Statement, A_Path, and An_Exception_Handler elements.

18.1 function Label_Names

```
function Label_Names (Statement : in Asis.Statement)
                     return Asis.Defining_Name_List;
```

Statement specifies the statement to query.

Returns label_statement_identifier elements (A_Defining_Name elements) that define the labels attached to the statement, in their order of appearance.

Returns a Nil_Element_List if there are no labels attached to the statement.

The Enclosing_Element of the A_Defining_Name elements is the statement.

Statement expects an element that has the following Element_Kinds:

A_Statement

Raises ASIS_Inappropriate_Element with a Status of Value_Error for any element that does not have one of these expected kinds.

Returns a list of elements that each have the following Defining_Name_Kinds:

A_Defining_Identifier

18.2 function Assignment_Variable_Name

```
function Assignment_Variable_Name (Statement : in Asis.Statement)
                                   return Asis.Name;
```

Statement specifies the assignment statement to query.

Returns the expression that names the left hand side of the assignment.

Statement expects an element that has the following Statement_Kinds:

An_Assignment_Statement

Raises ASIS_Inappropriate_Element with a Status of Value_Error for any element that does not have one of these expected kinds.

Returns an element that has the following Element_Kinds:

An_Expression

18.3 function Assignment_Expression

```
function Assignment_Expression (Statement : in Asis.Statement)
                                return Asis.Expression;
```

Statement specifies the assignment statement to query

Returns the expression from the right hand side of the assignment.

Statement expects an element that has the following Statement_Kinds:

An_Assignment_Statement

Raises ASIS_Inappropriate_Element with a Status of Value_Error for any element that does not have one of these expected kinds.

Returns an element that has the following Element_Kinds:

An_Expression

18.4 function Statement_Paths

```
function Statement_Paths (Statement : in Asis.Statement;
                          Include_Pragmas : in Boolean := False)
                           return Asis.Path_List;
```

Statement specifies the statement to query. Include_Pragmas specifies whether pragmas are to be returned.

Returns a list of the execution paths of the statement, in their order of appearance.

The only pragmas returned are those preceding the first alternative in a case statement.

Statement expects an element that has one of the following Statement_Kinds:

An_If_Statement
A_Case_Statement
A_Selective_Accept_Statement
A_Timed_Entry_Call_Statement
A_Conditional_Entry_Call_Statement
An_Asynchronous_Select_Statement

Raises ASIS_Inappropriate_Element with a Status of Value_Error for any element that does not have one of these expected kinds.

Returns a list of elements that each have one of the following Element_Kinds:

A_Path
A_Pragma

18.5 function Condition_Expression

```
function Condition_Expression (Path : in Asis.Path)
                              return Asis.Expression;
```

Path specifies the execution path to query.

Returns the condition expression for an **if** path or an **elsif** path.

Path expects an element that has one of the following Path_Kinds:

An_If_Path
An_Elsif_Path

Raises ASIS_Inappropriate_Element with a Status of Value_Error for any element that does not have one of these expected kinds.

Returns an element that has the following Element_Kinds:

An_Expression

18.6 function Sequence_Of_Statements

```
function Sequence_Of_Statements (Path            : in Asis.Path;
                                 Include_Pragmas : in Boolean := False)
                                 return Asis.Statement_List;
```

Path specifies the execution path to query. Include_Pragmas specifies whether pragmas are to be returned.

Returns a list of the statements and pragmas from an execution path, in their order of appearance.

Path expects an element that has the following Element_Kinds:

A_Path

Raises ASIS_Inappropriate_Element with a Status of Value_Error for any element that does not have one of these expected kinds.

Returns a list of elements that each have one of the following Element_Kinds:

A_Statement
A_Pragma

18.7 function Case_Expression

```
function Case_Expression (Statement : in Asis.Statement)
                          return Asis.Expression;
```

Statement specifies the case statement to query.

Returns the expression of the case statement that determines which execution path is taken.

Statement expects an element that has the following Statement_Kinds:

A_Case_Statement

Raises ASIS_Inappropriate_Element with a Status of Value_Error for any element that does not have one of these expected kinds.

Returns an element that has the following Element_Kinds:

An_Expression

18.8 function Case_Statement_Alternative_Choices

```
function Case_Statement_Alternative_Choices (Path : in Asis.Path)
                                             return Asis.Element_List;
```

Path specifies the case_statement_alternative execution path to query.

Returns a list of the "**when** <choice> | <choice>" elements, in their order of appearance.

Path expects an element that has the following Path_Kinds:

A_Case_Path

Raises ASIS_Inappropriate_Element with a Status of Value_Error for any element that does not have one of these expected kinds.

Returns a list of elements that each have one of the following Element_Kinds:

An_Expression
A_Definition that has one of the following Definition_Kinds:
  A_Discrete_Range
  An_Others_Choice

18.9 function Statement_Identifier

```
function Statement_Identifier (Statement : in Asis.Statement)
                               return Asis.Defining_Name;
```

Statement specifies the statement to query.

Returns the identifier for the loop_statement or block_statement.

Returns a Nil_Element if the loop has no identifier.

The Enclosing_Element of the name is the statement.

Statement expects an element that has one of the following Statement_Kinds:

A_Loop_Statement
A_While_Loop_Statement
A_For_Loop_Statement
A_Block_Statement

Raises ASIS_Inappropriate_Element with a Status of Value_Error for any element that does not have one of these expected kinds.

Returns an element that has one of the following Defining_Name_Kinds:

Not_A_Defining_Name
A_Defining_Identifier

18.10 function Is_Name_Repeated (statement)

```
function Is_Name_Repeated (Statement : in Asis.Statement) return Boolean;
```

Statement specifies the statement to query.

Returns True if the name of the accept, loop, or block is repeated after the end of the statement. Always returns True for loop or block statements since the name is required.

Returns False for any unexpected Element.

Statement expects an element that of one of the following Statement_Kinds:

A_Block_Statement
A_Loop_Statement
An_Accept_Statement

18.11 function While_Condition

```
function While_Condition (Statement : in Asis.Statement)
                         return Asis.Expression;
```

Statement specifies the loop statement to query.

Returns the condition expression associated with the while loop.

Statement expects an element that has the following Statement_Kinds:

A_While_Loop_Statement

Raises ASIS_Inappropriate_Element with a Status of Value_Error for any element that does not have one of these expected kinds.

Returns an element that has the following Element_Kinds:

An_Expression

18.12 function For_Loop_Parameter_Specification

```
function For_Loop_Parameter_Specification (Statement : in Asis.Statement)
                                           return Asis.Declaration;
```

Statement specifies the loop statement to query.

Returns the declaration of the A_Loop_Parameter_Specification.

Statement expects an element that has the following Statement_Kinds:

A_For_Loop_Statement

Raises ASIS_Inappropriate_Element with a Status of Value_Error for any element that does not have one of these expected kinds.

Returns an element that has the following Declaration_Kinds:

A_Loop_Parameter_Specification

18.13 function Loop_Statements

```
function Loop_Statements (Statement       : in Asis.Statement;
                          Include_Pragmas : in Boolean := False)
                          return Asis.Statement_List;
```

Statement specifies the loop statement to query. Include_Pragmas specifies whether pragmas are to be returned.

Returns the sequence_of_statements and any pragmas from the loop_statement, in their order of appearance.

Statement expects an element that has one of the following Statement_Kinds:

A_Loop_Statement
A_While_Loop_Statement
A_For_Loop_Statement

Raises ASIS_Inappropriate_Element with a Status of Value_Error for any element that does not have one of these expected kinds.

Returns a list of elements that each have one of the following Element_Kinds:

A_Pragma
A_Statement

18.14 function Is_Declare_Block

```
function Is_Declare_Block (Statement : in Asis.Statement) return Boolean;
```

Statement specifies the statement to query.

Returns True if the statement is a block_statement and it was created with the use of the **declare** reserved word. The presence or absence of any declarative_item elements is not relevant.

Returns False if the **declare** reserved word does not appear in the block_statement, or for any unexpected Element.

Statement expects an element that has the following Statement_Kinds:

A_Block_Statement

18.15 function Block_Declarative_Items

```
function Block_Declarative_Items
        (Statement       : in Asis.Statement;
            Include_Pragmas : in Boolean := False)
            return Asis.Declarative_Item_List;
```

Statement specifies the block statement to query. Include_Pragmas specifies whether pragmas are to be returned.

Returns a list of the declarations, aspect_clause elements, pragmas, and use_clause elements in the declarative_part of the block_statement, in their order of appearance.

Returns a Nil_Element_List if there are no declarative items.

Statement expects an element that has the following Statement_Kinds:

A_Block_Statement

Raises ASIS_Inappropriate_Element with a Status of Value_Error for any element that does not have one of these expected kinds.

Returns a list of elements that each have one of the following Element_Kinds:

A_Declaration
A_Pragma
A_Clause

18.16 function Block_Statements

```
function Block_Statements (Statement       : in Asis.Statement;
                           Include_Pragmas : in Boolean := False)
                           return Asis.Statement_List;
```

Statement specifies the block statement to query. Include_Pragmas specifies whether pragmas are to be returned.

Returns a list of the statements and pragmas for the block_statement, in their order of appearance.

Returns a Nil_Element_List if there are no statements or pragmas. This can only occur for a block_statement obtained from the obsolescent query Body_Block_Statement when its argument is a package_body that has no sequence_of_statements.

Statement expects an element that has the following Statement_Kinds:

A_Block_Statement

Raises ASIS_Inappropriate_Element with a Status of Value_Error for any element that does not have one of these expected kinds.

Returns a list of elements that each have one of following Element_Kinds:

A_Pragma
A_Statement

18.17 function Block_Exception_Handlers

```
function Block_Exception_Handlers (Statement : in Asis.Statement;
                                   Include_Pragmas : in Boolean := False)
                                   return Asis.Exception_Handler_List;
```

Statement specifies the block statement to query. Include_Pragmas specifies whether pragmas are to be returned.

Returns a list of the exception_handler elements of the block_statement, in their order of appearance.

The only pragmas returned are those following the reserved word **exception** and preceding the reserved word **when** of first exception handler.

Returns a Nil_Element_List if there are no exception_handler elements.

Statement expects an element that has the following Statement_Kinds:

A_Block_Statement

Raises ASIS_Inappropriate_Element with a Status of Value_Error for any element that does not have one of these expected kinds.

Returns a list of elements that each have one of the following Element_Kinds:

An_Exception_Handler
A_Pragma

18.18 function Exit_Loop_Name

```
function Exit_Loop_Name (Statement : in Asis.Statement)
                         return Asis.Name;
```

Statement specifies the exit statement to query.

Returns the name of the exited loop.

Returns a Nil_Element if no loop name is present.

Statement expects an element that has the following Statement_Kinds:

An_Exit_Statement

Raises ASIS_Inappropriate_Element with a Status of Value_Error for any element that does not have one of these expected kinds.

Returns an element that has one of the following Expression_Kinds:

Not_An_Expression
An_Identifier
A_Selected_Component

18.19 function Exit_Condition

```
function Exit_Condition (Statement : in Asis.Statement)
                         return Asis.Expression;
```

Statement specifies the exit statement to query.

Returns the **when** condition of the exit statement.

Returns a Nil_Element if no condition is present.

Statement expects an element that has the following Statement_Kinds:

An_Exit_Statement

Raises ASIS_Inappropriate_Element with a Status of Value_Error for any element that does not have one of these expected kinds.

Returns an element that has one of the following Element_Kinds:

Not_An_Element
An_Expression

18.20 function Corresponding_Loop_Exited

```
function Corresponding_Loop_Exited (Statement : in Asis.Statement)
                                    return Asis.Statement;
```

Statement specifies the exit statement to query.

Returns the loop statement exited by the exit statement.

Statement expects an element that has the following Statement_Kinds:

An_Exit_Statement

Raises ASIS_Inappropriate_Element with a Status of Value_Error for any element that does not have one of these expected kinds.

Returns an element that has one of the following Element_Kinds:

A_Loop_Statement
A_While_Loop_Statement
A_For_Loop_Statement

18.21 function Return_Expression

```
function Return_Expression (Statement : in Asis.Statement)
                           return Asis.Expression;
```

Statement specifies the return statement to query.

Returns the expression in the return statement.

Returns a Nil_Element if no expression is present.

Statement expects an element that has the following Statement_Kinds:

A_Return_Statement

Raises ASIS_Inappropriate_Element with a Status of Value_Error for any element that does not have one of these expected kinds.

Returns an element that has one of the following Element_Kinds:

Not_An_Element
An_Expression

18.22 function Return_Object_Specification

```
function Return_Object_Specification
     (Statement : in Asis.Statement)
     return Asis.Declaration;
```

Statement specifies the extended return statement to query.

Returns the specification of the return object.

Statement expects an element that has the following Statement_Kinds:

An_Extended_Return_Statement

Raises ASIS_Inappropriate_Element with a Status of Value_Error for any element that does not have one of these expected kinds.

Returns an element that has the following Declaration_Kinds:

A_Return_Object_Specification

18.23 function Extended_Return_Statements

```
function Extended_Return_Statements
     (Statement       : in Asis.Statement;
      Include_Pragmas : in Boolean := False)
      return Asis.Statement_List;
```

Statement specifies the extended return statement to query. Include_Pragmas specifies whether pragmas are to be returned.

Returns a list of the statements and pragmas from the extended return statement, in their order of appearance.

Returns a Nil_Element_List if the argument extended return statement does not include handled_sequence_of_statements.

Statement expects an element that has the following Statement_Kinds:

An_Extended_Return_Statement

Raises ASIS_Inappropriate_Element with a Status of Value_Error for any element that does not have one of these expected kinds.

Returns a list of elements that each is one of the following Element_Kinds:

A_Statement
A_Pragma

18.24 function Extended_Return_Exception_Handlers

```
function Extended_Return_Exception_Handlers
     (Statement       : in Asis.Statement;
      Include_Pragmas : in Boolean := False)
      return Asis.Exception_Handler_List;
```

Statement specifies the extended return statement to query. Include_Pragmas specifies whether pragmas are to be returned.

Returns a list of the exception_handler elements of the extended return statement, in their order of appearance.

The only pragmas returned are those following the reserved word "exception" and preceding the reserved word "when" of first exception handler.

Returns a Nil_Element_List if there are no exception_handler elements.

Statement expects an element that has the following Statement_Kinds:

An_Extended_Return_Statement

Raises ASIS_Inappropriate_Element with a Status of Value_Error for any element that does not have one of these expected kinds.

Returns a list of elements that each have one of the following Element_Kinds:

An_Exception_Handler
A_Pragma

18.25 function Goto_Label

```
function Goto_Label (Statement : in Asis.Statement)
                     return Asis.Name;
```

Statement specifies the goto statement to query.

Returns the expression reference for the label, as specified by the goto statement.

Statement expects an element that has the following Statement_Kinds:

A_Goto_Statement

Raises ASIS_Inappropriate_Element with a Status of Value_Error for any element that does not have one of these expected kinds.

Returns an element that has the following Expression_Kinds:

An_Identifier

18.26 function Corresponding_Destination_Statement

```
function Corresponding_Destination_Statement
   (Statement : in Asis.Statement)
    return Asis.Statement;
```

Statement specifies the goto statement to query.

Returns the target statement specified by the goto statement.

Statement expects an element that has the following Statement_Kinds:

A_Goto_Statement

Raises ASIS_Inappropriate_Element with a Status of Value_Error for any element that does not have one of these expected kinds.

Returns an element that has the following Element_Kinds:

A_Statement

18.27 function Called_Name

```
function Called_Name (Statement : in Asis.Statement)
                     return Asis.Name;
```

Statement specifies the procedure call or entry call statement to query.

Returns the name of the called procedure or entry. The name of an entry family takes the form of An_Indexed_Component.

Statement expects an element that has one of the following Statement_Kinds:

An_Entry_Call_Statement
A_Procedure_Call_Statement

Raises ASIS_Inappropriate_Element with a Status of Value_Error for any element that does not have one of these expected kinds.

Returns an element that has the following Element_Kinds:

An_Expression

18.28 function Corresponding_Called_Entity

```
function Corresponding_Called_Entity (Statement : in Asis.Statement)
                                      return Asis.Declaration;
```

Statement specifies the procedure_call_statement or entry_call_statement to query.

Returns the declaration of the procedure or entry denoted by the call.

Returns a Nil_Element if the:

- prefix of the call denotes an implicit or explicit dereference of an access to a procedure value, or
- Statement is a to a dispatching operation of a tagged type which is not statically determined.

If the *procedure*_prefix or *procedure*_name denotes an attribute_reference, and if the corresponding attribute is (re)defined by an attribute definition clause, an implementation is encouraged, but not required, to return the definition of the corresponding subprogram whose name is used after **use** in this attribute definition clause. If an implementation cannot return such a subprogram definition, a Nil_Element should be returned. For an

attribute reference which is not (re)defined by an attribute definition clause, a Nil_Element should be returned.

Statement expects an element that has one of the following Statement_Kinds:

An_Entry_Call_Statement
A_Procedure_Call_Statement

Raises ASIS_Inappropriate_Element with a Status of Value_Error for any element that does not have one of these expected kinds.

Returns an element that has one of the following Declaration_Kinds:

Not_A_Declaration
A_Procedure_Declaration
A_Procedure_Body_Declaration
A_Procedure_Body_Stub
A_Procedure_Renaming_Declaration
A_Procedure_Instantiation
A_Formal_Procedure_Declaration
An_Entry_Declaration
A_Generic_Procedure_Declaration

Implementation Permissions

An implementation may choose to return any part of multi-part declarations and definitions. Multi-part declaration/definitions can occur for:

- Subprogram specification in package specification, package body, and subunits (**is separate**);
- Entries in package specification, package body, and subunits (**is separate**);
- Private type and full type declarations;
- Incomplete type and full type declarations; and
- Deferred constant and full constant declarations.

No guarantee is made that the element will be the first part or that the determination will be made due to any visibility rules. An application should make its own analysis for each case based on which part is returned.

18.29 function Call_Statement_Parameters

```
function Call_Statement_Parameters (Statement  : in Asis.Statement;
                                    Normalized : in Boolean := False)
                                    return Asis.Association_List;
```

Statement specifies the procedure_call_statement or entry_call_statement to query. Normalized specifies whether the normalized form is desired.

Returns a list of parameter_association elements of the call.

Returns a Nil_Element_List if there are no parameter_association elements.

An unnormalized list contains only explicit associations ordered as they appear in the program text. Each unnormalized association has an optional formal_parameter_selector_name and an explicit_actual_parameter component.

A normalized list contains artificial associations representing all explicit and default associations. It has a length equal to the number of parameter_specification elements of the formal_part of the parameter_and_result_profile. The order of normalized associations matches the order of parameter_specification elements.

Each normalized association represents a one-to-one mapping of a parameter_specification element to the explicit or default expression. A normalized association has one A_Defining_Name component that denotes the parameter_specification, and one An_Expression component that is either the explicit_actual_parameter, a default_expression, or when the call uses a prefixed view of the subprogram, the prefix of the call.

If the prefix of the call denotes an implicit or explicit dereference of an access to a procedure value, normalized associations are constructed on the basis of the formal_part of the parameter_profile from the corresponding access_to_subprogram definition.

Returns Nil_Element for normalized associations in the case where the called procedure can be determined only dynamically (dispatching calls). ASIS cannot produce any meaningful result in this case.

The exception ASIS_Inappropriate_Element is raised when the procedure call is an attribute reference and Is_Normalized is True.

Statement expects an element that has one of the following Statement_Kinds:

An_Entry_Call_Statement
A_Procedure_Call_Statement

Raises ASIS_Inappropriate_Element with a Status of Value_Error for any element that does not have one of these expected kinds.

Returns a list of elements that each have the following Element_Kinds:

A_Parameter_Association

Implementation Requirements

Normalized associations are Is_Normalized and Is_Part_Of_Implicit. Normalized associations provided by default are Is_Defaulted_Association. Normalized associations are never Is_Equal to unnormalized associations.

18.30 function Accept_Entry_Index

```
function Accept_Entry_Index (Statement : in Asis.Statement)
                             return Asis.Expression;
```

Statement specifies the accept statement to query.

Returns the entry index expression in the accept statement.

Returns a Nil_Element if the statement has no explicit entry index,

Statement expects an element that has the following Statement_Kinds:

An_Accept_Statement

Raises ASIS_Inappropriate_Element with a Status of Value_Error for any element that does not have one of these expected kinds.

Returns an element that has one of the following Element_Kinds:

Not_An_Element
An_Expression

18.31 function Accept_Entry_Direct_Name

```
function Accept_Entry_Direct_Name (Statement : in Asis.Statement)
                                   return Asis.Name;
```

Statement specifies the accept statement to query.

Returns the direct name of the entry. The name follows the reserved word **accept**.

Statement expects an element that has the following Statement_Kinds:

An_Accept_Statement

Raises ASIS_Inappropriate_Element with a Status of Value_Error for any element that does not have one of these expected kinds.

Returns an element that has the following Expression_Kinds:

An_Identifier

18.32 function Accept_Parameters

```
function Accept_Parameters (Statement : in Asis.Statement)
                            return Asis.Parameter_Specification_List;
```

Statement specifies the accept statement to query.

Returns a list of parameter specifications in the formal part of the accept statement, in their order of appearance.

Returns a Nil_Element_List if the accept_statement has no parameters.

Results of this query may vary across ASIS implementations. Some implementations normalize all multiple name parameter specifications into an equivalent sequence of corresponding single name parameter specifications. See Ada Standard 3.3.1(7).

Statement expects an element that has the following Statement_Kinds:

An_Accept_Statement

Raises ASIS_Inappropriate_Element with a Status of Value_Error for any element that does not have one of these expected kinds.

Returns a list of elements that each have the following Declaration_Kinds:

A_Parameter_Specification

18.33 function Accept_Body_Statements

```
function Accept_Body_Statements (Statement       : in Asis.Statement;
                                 Include_Pragmas : in Boolean := False)
                                 return Asis.Statement_List;
```

Statement specifies the accept statement to query. Include_Pragmas specifies whether pragmas are to be returned.

Returns the list of statements and pragmas from the body of the accept statement, in their order of appearance.

Statement expects an element that has the following Statement_Kinds:

An_Accept_Statement

Raises ASIS_Inappropriate_Element with a Status of Value_Error for any element that does not have one of these expected kinds.

Returns a list of elements that each have one of the following Element_Kinds:

A_Pragma
A_Statement

18.34 function Accept_Body_Exception_Handlers

```
function Accept_Body_Exception_Handlers
             (Statement       : in Asis.Statement;
              Include_Pragmas : in Boolean := False)
              return Asis.Statement_List;
```

Statement specifies the accept statement to query. Include_Pragmas specifies whether pragmas are to be returned.

Returns the list of exception handlers and pragmas from the body of the accept statement, in their order of appearance.

Statement expects an element that has the following Statement_Kinds:

An_Accept_Statement

Raises ASIS_Inappropriate_Element with a Status of Value_Error for any element that does not have one of these expected kinds.

Returns a list of elements that each have one of the following Element_Kinds:

A_Pragma
An_Exception_Handler

18.35 function Corresponding_Entry

```
function Corresponding_Entry (Statement : in Asis.Statement)
                              return Asis.Declaration;
```

Statement specifies the accept statement to query.

Returns the declaration of the entry accepted in this statement.

Statement expects an element that has the following Statement_Kinds:

An_Accept_Statement

Raises ASIS_Inappropriate_Element with a Status of Value_Error for any element that does not have one of these expected kinds.

Returns an element that has the following Element_Kinds:

An_Entry_Declaration

18.36 function Requeue_Entry_Name

```
function Requeue_Entry_Name (Statement : in Asis.Statement)
                             return Asis.Name;
```

Statement specifies the requeue statement to query.

Returns the name of the entry requeued by the statement. The name follows the reserved word **requeue**.

Statement expects an element that has one of the following Statement_Kinds:

A_Requeue_Statement
A_Requeue_Statement_With_Abort

Raises ASIS_Inappropriate_Element with a Status of Value_Error for any element that does not have one of these expected kinds.

Returns an element that has the following Element_Kinds:

An_Expression

18.37 function Delay_Expression

```
function Delay_Expression (Statement : in Asis.Statement)
                           return Asis.Expression;
```

Statement specifies the delay statement to query.

Returns the expression for the duration of the delay.

Statement expects an element that has one of the following Statement_Kinds:

A_Delay_Until_Statement
A_Delay_Relative_Statement

Raises ASIS_Inappropriate_Element with a Status of Value_Error for any element that does not have one of these expected kinds.

Returns an element that has the following Element_Kinds:

An_Expression

18.38 function Guard

```
function Guard (Path : in Asis.Path)
                return Asis.Expression;
```

Path specifies the select statement execution path to query.

Returns the conditional expression guard for the path.

Returns a Nil_Element if there is no guard, or if the path is from a timed_entry_call, a conditional_entry_call, or an asynchronous_select statement where a guard is not legal.

Path expects an element that has one of the following Path_Kinds:

A_Select_Path
An_Or_Path

Raises ASIS_Inappropriate_Element with a Status of Value_Error for any element that does not have one of these expected kinds.

Returns an element that has one of the following Element_Kinds:

Not_An_Element
An_Expression

18.39 function Aborted_Tasks

```
function Aborted_Tasks (Statement : in Asis.Statement)
                        return Asis.Name_List;
```

Statement specifies the abort statement to query.

Returns a list of the task names from the **abort** statement, in their order of appearance.

Statement expects an element that has the following Statement_Kinds:

An_Abort_Statement

Raises ASIS_Inappropriate_Element with a Status of Value_Error for any element that does not have one of these expected kinds.

Returns a list of elements that each have the following Element_Kinds:

An_Expression

18.40 function Choice_Parameter_Specification

```
function Choice_Parameter_Specification
   (Handler : in Asis.Exception_Handler)
    return Asis.Declaration;
```

Handler specifies the exception handler to query.

Returns the choice parameter specification following the reserved word **when** in the exception handler.

Returns a Nil_Element if there is no explicit choice parameter.

Handler expects an element that has the following Element_Kinds:

An_Exception_Handler

Raises ASIS_Inappropriate_Element with a Status of Value_Error for any element that does not have one of these expected kinds.

Returns an element that has one of the following Declaration_Kinds:

Not_A_Declaration
A_Choice_Parameter_Specification

18.41 function Exception_Choices

```
function Exception_Choices (Handler : in Asis.Exception_Handler)
                           return Asis.Element_List;
```

Handler specifies the exception handler to query.

Returns a list of the "**when** <choice> | <choice>" elements, in their order of appearance. Choices are either the exception name expression or an others choice.

Handler expects an element that has the following Element_Kinds:

An_Exception_Handler

Raises ASIS_Inappropriate_Element with a Status of Value_Error for any element that does not have one of these expected kinds.

Returns a list of elements that each have one of the following Expression_Kinds:

An_Identifier
A_Selected_Component

or an element that has the following Definition_Kinds:

An_Others_Choice

18.42 function Handler_Statements

```
function Handler_Statements (Handler         : in Asis.Exception_Handler;
                             Include_Pragmas : in Boolean := False)
                             return Asis.Statement_List;
```

Handler specifies the exception handler to query. Include_Pragmas specifies whether pragmas are to be returned.

Returns the list of statements and pragmas from the body of the exception handler, in their order of appearance.

Handler expects an element that has the following Element_Kinds:

An_Exception_Handler

Raises ASIS_Inappropriate_Element with a Status of Value_Error for any element that does not have one of these expected kinds.

Returns a list of elements that each have one of the following Element_Kinds:

A_Pragma
A_Statement

18.43 function Raised_Exception

```
function Raised_Exception (Statement : in Asis.Statement)
                           return Asis.Name;
```

Statement specifies the raise statement to query.

Returns the expression that names the raised exception.

Returns a Nil_Element if there is no explicitly named exception.

Statement expects an element that has the following Statement_Kinds:

A_Raise_Statement

Raises ASIS_Inappropriate_Element with a Status of Value_Error for any element that does not have one of these expected kinds.

Returns an element that has one of the following Expression_Kinds:

Not_An_Expression
An_Identifier
A_Selected_Component

18.44 function Raise_Statement_Message

```
function Raise_Statement_Message (Statement : in Asis.Statement)
                           return Asis.Expression;
```

Statement specifies the raise statement to query.

Returns the string expression that is associated with the raised exception and follows the **with** keyword in the raise statement.

Returns a Nil_Element if there is no string expression.

Statement expects an element that has the following Statement_Kinds:

A_Raise_Statement

Raises ASIS_Inappropriate_Element with a Status of Value_Error for any element that does not have one of these expected kinds.

Returns an element that has one of the following Element_Kinds:

Not_An_Element
An_Expression

18.45 function Qualified_Expression

```
function Qualified_Expression (Statement : in Asis.Statement)
                               return Asis.Expression;
```

Statement specifies the code statement to query.

Returns the qualified aggregate expression representing the code statement.

Statement expects an element that has the following Statement_Kinds:

A_Code_Statement

Raises ASIS_Inappropriate_Element with a Status of Value_Error for any element that does not have one of these expected kinds.

Returns an element that has the following Expression_Kinds:

A_Qualified_Expression

18.46 function Is_Dispatching_Call

```
function Is_Dispatching_Call (Call : in Asis.Element) return Boolean;
```

Call specifies the element to query.

Returns True if the controlling tag of Call is dynamically determined.

This function shall always return False when pragma Restrictions(No_Dispatch) applies.

Returns False for any unexpected Element.

Call expects an element that has the following Expression_Kinds:

A_Function_Call

or Call expects an element that has the following Statement_Kinds:

A_Procedure_Call_Statement

18.47 function Is_Call_On_Dispatching_Operation

```
function Is_Call_On_Dispatching_Operation (Call : in Asis.Element)
                                           return Boolean;
```

Call specifies the element to query.

Returns True if the name or prefix of Call denotes the declaration of a primitive operation of a tagged type.

Returns False for any unexpected Element.

Call expects an element that has one of the following Element_Kinds:

A_Function_Call
A_Procedure_Call_Statement

Section 19: package Asis.Clauses

The library package Asis.Clauses shall exist. The package shall provide interfaces equivalent to those described in the following subclauses.

This package encapsulates a set of queries that operate on A_Clause elements.

19.1 function Clause_Names

```
function Clause_Names (Clause : in Asis.Element)
                       return Asis.Name_List;
```

Clause specifies the with_clause or use_clause to query.

Returns a list of the names that appear in the given clause. The names in the list should be in their order of appearance in the original clauses from the compilation text.

Results of this query may vary across ASIS implementations. Some implementations normalize all clauses containing multiple names into an equivalent sequence of corresponding single clauses. Similarly, an implementation may keep a name only once even though that name can appear more than once in a clause.

Clause expects an element that has one of the following Element_Kinds:

A_Use_Package_Clause
A_Use_Type_Clause
A_With_Clause

Raises ASIS_Inappropriate_Element with a Status of Value_Error for any element that does not have one of these expected kinds.

Returns a list of elements that each has one of the following Expression_Kinds:

An_Identifier
A_Selected_Component
An_Attribute_Reference

19.2 function Aspect_Clause_Name

```
function Aspect_Clause_Name (Clause : in Asis.Clause)
                                     return Asis.Name;
```

Clause specifies the aspect_clause or component_clause to query.

Returns the direct_name expression following the reserved word **for**.

Clause expects an element that has one of the following Clause_Kinds:

An_Aspect_Clause
A_Component_Clause

Raises ASIS_Inappropriate_Element with a Status of Value_Error for any element that does not have one of these expected kinds.

Returns an element that has one of the following Expression_Kinds:

An_Identifier
An_Attribute_Reference

19.3 function Aspect_Clause_Expression

```
function Aspect_Clause_Expression
            (Clause : in Asis.Aspect_Clause)
             return Asis.Expression;
```

Clause specifies the aspect_clause to query.

Returns the expression following the reserved word **use** or the reserved words **use at**.

Clause expects an element that has one of the following Aspect_Clause_Kinds:

An_Attribute_Definition_Clause
An_Enumeration_Representation_Clause
An_At_Clause

Raises ASIS_Inappropriate_Element with a Status of Value_Error for any element that does not have one of these expected kinds.

Returns an element that has the following Element_Kinds:

An_Expression

19.4 function Mod_Clause_Expression

```
function Mod_Clause_Expression (Clause : in Asis.Aspect_Clause)
                               return Asis.Expression;
```

Clause specifies the record representation clause to query.

Returns the static_expression appearing after the reserved words **at mod**.

Returns a Nil_Element if a mod_clause is not present.

Clause expects an element that has the following Aspect_Clause_Kinds:

A_Record_Representation_Clause

Raises ASIS_Inappropriate_Element with a Status of Value_Error for any element that does not have one of these expected kinds.

Returns an element that has one of the following Element_Kinds:

Not_An_Element
An_Expression

19.5 function Component_Clauses

```
function Component_Clauses (Clause : in Asis.Aspect_Clause;
                            Include_Pragmas : in Boolean := False)
                            return Asis.Component_Clause_List;
```

Clause specifies the record representation clause to query. Include_Pragmas specifies whether pragmas are to be returned.

Returns the component_clause and pragma elements from the record_representation_clause, in their order of appearance.

Returns a Nil_Element_List if the record_representation_clause has no component_clause or pragma elements.

Clause expects an element that has one of the following Aspect_Clause_Kinds:

A_Record_Representation_Clause

Raises ASIS_Inappropriate_Element with a Status of Value_Error for any element that does not have one of these expected kinds.

Returns a list of elements that each has one of the following Element_Kinds:

A_Clause — the returned element also has Clause_Kinds:
A_Component_Clause
A_Pragma

19.6 function Component_Clause_Position

```
function Component_Clause_Position (Clause : in Asis.Component_Clause)
                                    return Asis.Expression;
```

Clause specifies the component_clause to query.

Returns the position expression for the component_clause.

Clause expects an element that has the following Clause_Kinds:

A_Component_Clause

Raises ASIS_Inappropriate_Element with a Status of Value_Error for any element that does not have one of these expected kinds.

Returns an element that has the following Element_Kinds:

An_Expression

19.7 function Component_Clause_Range

```
function Component_Clause_Range (Clause : in Asis.Component_Clause)
                                 return Asis.Discrete_Range;
```

Clause specifies the component_clause to query.

Returns the first_bit .. last_bit range for the component_clause.

Clause expects an element that has the following Clause_Kinds:

A_Component_Clause

Raises ASIS_Inappropriate_Element with a Status of Value_Error for any element that does not have one of these expected kinds.

Returns an element that has the following Discrete_Range_Kinds:

A_Discrete_Simple_Expression_Range

Section 20: package Asis.Text

The library package Asis.Text shall exist. The package shall provide interfaces equivalent to those described in the following subclauses.

This package encapsulates a set of operations to access the text of ASIS Elements. It assumes no knowledge of the existence, location, or form of the program text.

The text of a program consists of the texts of one or more compilations. The text of each compilation is a sequence of separate lexical elements. Each lexical element is either a delimiter, an identifier (which can be a reserved word), a numeric literal, a character literal, a string literal, blank space, or a comment.

Each ASIS Element has a text image whose value is the series of characters contained by the text span of the Element. The text span covers all the characters from the first character of the Element through the last character of the Element over some range of lines.

General Usage Rules:

Line lists can be indexed to obtain individual lines. The bounds of each list correspond to the lines with those same numbers from the compilation text.

Any Asis.Text query may raise ASIS_Failed with a Status of Text_Error if the program text cannot be located or retrieved for any reason such as renaming, deletion, corruption, or moving of the text.

20.1 type Line

The type *Line* represents text fragments from a compilation. ASIS Lines are representations of the compilation text.

```
type Line is private;
Nil_Line  : constant Line;

function "=" (Left  : in Line;
              Right : in Line)
              return Boolean is abstract;
```

Nil_Line is the value of a default-initialized Line object.

20.2 subtypes Line_Number and Line_Number_Positive

Line_Number is a numeric subtype that allows each ASIS implementation to place constraints on the upper bound for Line_List elements and compilation unit size.

The upper bound of Line_Number (Maximum_Line_Number) is the only allowed variation for these declarations.

Line_Number = 0 is reserved to act as an "invalid" Line_Number value. No unit text line will ever have a Line_Number of zero.

```
Maximum_Line_Number : constant ASIS_Natural :=
    Implementation_Defined_Integer_Constant;

subtype Line_Number is ASIS_Natural range 0 .. Maximum_Line_Number;

subtype Line_Number_Positive is Line_Number range 1 .. Maximum_Line_Number;
```

20.3 type Line_List

```
type Line_List is array (Line_Number_Positive range <>) of Line;
Nil_Line_List : constant Line_List;
```

Nil_Line_List is the value of a Line_List containing no elements.

20.4 subtypes Character_Position and Character_Position_Positive

Character_Position is a numeric subtype that allows each ASIS implementation to place constraints on the upper bound for Character_Position and for compilation unit line lengths.

The upper bound of Character_Position (Maximum_Line_Length) is the only allowed variation for these declarations.

Character_Position = 0 is reserved to act as an "invalid" Character_Position value. No unit text line will ever have a character in position zero.

```
Maximum_Line_Length : constant ASIS_Natural :=
   Implementation_Defined_Integer_Constant;

subtype Character_Position is ASIS_Natural range 0 .. Maximum_Line_Length;

subtype Character_Position_Positive is
   Character_Position range 1 .. Maximum_Line_Length;
```

20.5 type Span

A *single text position* is identified by a line number and a column number representing the text's location within the compilation unit.

The text of an element can span one or more lines. The textual Span of an element identifies the lower and upper bound of a span of text positions.

Spans and positions give client tools the option of accessing compilation unit text through the queries provided by this package, or, to access the text directly through the original compilation unit text file. Type Span facilitates the capture of comments before or after an element.

```
type Span is                                          -- Default is Nil_Span
   record
      First_Line   : Line_Number_Positive        := 1; -- 1..0 - empty
      First_Column : Character_Position_Positive := 1; -- 1..0 - empty
      Last_Line    : Line_Number                 := 0;
      Last_Column  : Character_Position          := 0;
   end record;

Nil_Span : constant Span := (First_Line   => 1,
                             First_Column => 1,
                             Last_Line    => 0,
                             Last_Column  => 0);
```

NOTE The original compilation unit text may or may not have existed in a "file", and any such file may or may not still exist. Ada Standard 10.1 specifies that the text of a compilation unit is submitted to a compiler. It does not specify that the text is stored in a "file", nor does it specify that the text of a compilation unit has any particular lifetime.

20.6 function First_Line_Number

```
function First_Line_Number (Element : in Asis.Element)
                           return Line_Number;
```

Element specifies the element to query.

Returns the first line number on which the text of the element resides.

Returns 0 if not Is_Text_Available(Element).

20.7 function Last_Line_Number

```
function Last_Line_Number (Element : in Asis.Element)
                           return Line_Number;
```

Element specifies the element to query.

Returns the last line number on which the text of the element resides.

Returns 0 if not Is_Text_Available(Element).

20.8 function Element_Span

```
function Element_Span (Element : in Asis.Element)
                       return Span;
```

Element specifies the element to query.

Returns the span of the given element.

Returns a Nil_Span if the text of a Compilation_Unit (Compilation) cannot be located for any reason.

20.9 function Compilation_Unit_Span

```
function Compilation_Unit_Span (Element : in Asis.Element)
                                return Span;
```

Element specifies the element to query.

Returns the span of the text comprising the enclosing compilation unit of the given element.

Returns a Nil_Span if the text of a Compilation_Unit (Compilation) cannot be located for any reason.

20.10 function Compilation_Span

```
function Compilation_Span (Element : in Asis.Element)
                           return Span;
```

Element specifies the element to query.

Returns the span of the text comprising the compilation to which the element belongs. The text span may include one or more compilation units.

Returns a Nil_Span if not Is_Text_Available(Element).

20.11 function Is_Nil (line)

```
function Is_Nil (Right : in Line)
                 return Boolean;
```

Right specifies the line to check.

Returns True if the argument is the Nil_Line otherwise returns False.

A Line from a Line_List obtained from any of the Lines functions will not be Is_Nil even if it has a length of zero.

20.12 function Is_Nil (line list)

```
function Is_Nil (Right : in Line_List)
                 return Boolean;
```

Right specifies the line list to check.

Returns True if the argument has a 'Length of zero otherwise returns False.

20.13 function Is_Nil (span)

```
function Is_Nil (Right : in Span)
                 return Boolean;
```

Right specifies the Span to check.

Returns True if the argument has a Nil_Span otherwise returns False.

20.14 function Is_Equal (lines)

```
function Is_Equal (Left  : in Line;
                   Right : in Line) return Boolean;
```

Left specifies the first of the two lines. Right specifies the second of the two lines.

Returns True if the two lines encompass the same text (have the same Span and are from the same compilation) otherwise returns False.

20.15 function Is_Identical (lines)

```
function Is_Identical (Left  : in Line;
                       Right : in Line) return Boolean;
```

Left specifies the first of the two lines. Right specifies the second of the two lines.

Returns True if the two lines encompass the same text (have the same Span and are from the same compilation) and are from the same Context otherwise returns False.

20.16 function Length

```
function Length (The_Line : in Line) return Character_Position;
```

The_Line specifies the line to query.

Returns the length of the line.

Raises ASIS_Inappropriate_Line if Is_Nil (The_Line).

20.17 function Lines (element)

```
function Lines (Element : in Asis.Element) return Line_List;
```

Element specifies the element to query.

Returns a list of lines covering the span of the given program element.

Returns a Nil_Line_List if the text of a Compilation containing a given Element cannot be located for any reason.

Line lists can be indexed to obtain individual lines. The bounds of each list correspond to the lines with those same numbers in the compilation text.

The first Line of the result contains text from the compilation starting at the First_Line/First_Column of Element's Span. The last Line of the result contains text from

the compilation ending at the Last_Line/Last_Column of the Element's Span. Text before or after those limits is not reflected in the returned list.

20.18 function Lines (element with span)

```
function Lines (Element  : in Asis.Element;
                The_Span : in Span) return Line_List;
```

Element specifies the element to query. The_Span specifies the textual span to return.

Returns a list of lines covering the given span from the compilation containing the given program element.

Returns a Nil_Line_List if the text of a Compilation containing a given Element cannot be located for any reason.

This operation can be used to access lines from text outside the span of an element, but still within the compilation. For example, lines containing preceding comments or lines between two elements.

Line lists can be indexed to obtain individual lines. The bounds of each list correspond to the lines with those same numbers in the compilation text.

The first Line of the result contains text from the compilation starting at line Span.First_Line and column Span.First_Column. The last Line of the result contains text from the compilation ending at line Span.Last_Line and column Span.Last_Column. Text before or after those limits is not reflected in the returned list.

Raises ASIS_Inappropriate_Line_Number if Is_Nil (The_Span). If The_Span defines a line whose number is outside the range of text lines that can be accessed through the Element, the implementation is encouraged, but not required to raise ASIS_Inappropriate_Line_Number.

20.19 function Lines (element with lines)

```
function Lines (Element    : in Asis.Element;
                First_Line : in Line_Number_Positive;
                Last_Line  : in Line_Number) return Line_List;
```

Element specifies the element to query. First_Line specifies the first line to return. Last_Line specifies the last line to return.

Returns a list of Lines covering the full text for each of the indicated lines from the compilation containing the given element. This operation can be used to access lines from text outside the span of an element, but still within the compilation.

Returns a Nil_Line_List if the text of a Compilation containing a given Element cannot be located for any reason.

Line lists can be indexed to obtain individual lines. The bounds of each list correspond to the lines with those same numbers in the compilation text.

Raises ASIS_Inappropriate_Line_Number if Last_Line is less than First_Line. If the span defines a line whose number is outside the range of text lines that can be accessed through the Element, the implementation is encouraged, but not required to raise ASIS_Inappropriate_Line_Number.

20.20 function Delimiter_Image

```
function Delimiter_Image return Wide_String;
```

Returns the string used as the delimiter separating individual lines of text within the program text image of an element. It is also used as the delimiter separating individual lines of strings returned by Debug_Image.

20.21 function Element_Image

```
function Element_Image (Element : in Asis.Element) return Program_Text;
```

Element specifies the element to query.

Returns a program text image of the element. The image of an element can span more than one line, in which case the program text returned by the function Delimiter_Image separates the individual lines. The bounds on the returned program text value are 1..N, N is as large as necessary.

Returns a null string if **not** Is_Text_Available(Element).

If an Element's Span begins at column position P, the returned program text will be padded at the beginning with P-1 white space characters (Ascii.' ' or Ascii.Ht). The first character of the Element's image will thus begin at character P of the returned program text. Due to the possible presence of Ascii.Ht characters, the "column" position of characters within the image might not be the same as their print-column position when the image is displayed on a screen or printed.

NOTE The image of a large element can exceed the range of Program_Text. In this case, the exception ASIS_Failed is raised with a Status of Capacity_Error. Use the Lines function to operate on the image of large elements.

20.22 function Line_Image

```
function Line_Image (The_Line : in Line) return Program_Text;
```

The_Line specifies the line to query.

Returns a program text image of the line. The image of a single lexical element can be sliced from the returned value using the first and last column character positions from the Span of the Element. The bounds on the returned program text are 1 .. Length(Line).

If the Line is the first line from the Lines result for an Element, it may represent only a portion of a line from the original compilation. If the span began at character position P, the first Line of its Lines result is padded at the beginning with P-1 white space characters (Ascii.' ' or Ascii.Ht). The first character of the image will thus begin at character P of the program text for the first Line. Due to the possible presence of Ascii.Ht characters, the "column" position of characters within the image may not be the same as their print-column position when the image is displayed or printed.

Similarly, if the Line is the last line from the Lines result for an Element, it may represent only a portion of a line from the original compilation. The program text image of such a Line is shorter than the line from compilation and will contain only the initial portion of that line.

Raises ASIS_Inappropriate_Line if Is_Nil (The_Line).

20.23 function Non_Comment_Image

```
function Non_Comment_Image (The_Line : in Line) return Program_Text;
```

The_Line specifies the line to query.

Returns a program text image of a Line up to, but excluding, any comment appearing in that Line.

The value returned is the same as that returned by the Image function, except that any hyphens ("--") that start a comment, and any characters that follow those hyphens, are dropped.

The bounds on the returned program text are 1..N, where N is one less than the column of any hyphens ("--") that start a comment on the line.

Raises ASIS_Inappropriate_Line if Is_Nil (The_Line).

20.24 function Comment_Image

```
function Comment_Image (The_Line : in Line) return Program_Text;
```

The_Line specifies the line to query.

Returns a program text image of any comment on that line, excluding any lexical elements preceding the comment.

The value returned is the same as that returned by the Image function, except that any program text prior to the two adjacent hyphens ("--") which start a comment is replaced by an equal number of spaces. If the hyphens began in column P of the Line, they will also begin in character position P of the returned program text.

A null string is returned if the line has no comment.

The bounds of the program text are 1..N, where N is as large as necessary.

Raises ASIS_Inappropriate_Line if Is_Nil (The_Line).

20.25 function Is_Text_Available

```
function Is_Text_Available (Element : in Asis.Element) return Boolean;
```

Element specifies the element to query.

Returns True if the implementation can return a valid text image for the given element.

Returns False for any Element that Is_Nil, Is_Part_Of_Implicit, or Is_Part_Of_Instance.

Returns False if the text of the element cannot be located for any reason such as renaming, deletion, or moving of text.

Implementation Requirements

An implementation shall make text available for all explicit elements.

20.26 function Debug_Image (line)

```
function Debug_Image (The_Line : in Line) return Wide_String;
```

The_Line specifies the line to convert.

Returns a string value containing implementation-defined debug information associated with the line.

Raises ASIS_Inappropriate_Line if Is_Nil (The_Line).

The return value uses Asis.Text.Delimiter_Image to separate the lines of multi-line results. The return value does not end with Asis.Text.Delimiter_Image.

These values are intended for two purposes. They are suitable for inclusion in problem reports sent to the ASIS implementor. They can be presumed to contain information useful when debugging the implementation itself. They are also suitable for use by the ASIS application when printing simple application debugging messages during application development. They are intended to be, to some worthwhile degree, intelligible to the user.

Section 21: package Asis.Ids

The library package Asis.Ids shall exist. The package shall provide interfaces equivalent to those described in the following subclauses.

Asis.Ids provides support for permanent unique Element "Identifiers" (Ids). An Id is an efficient way for a tool to reference an ASIS element. The Id is permanent from session to session provided that the Ada compilation environment is unchanged. This package encapsulates a set of operations and queries that implement the ASIS Id abstraction. An Id is a way of identifying a particular Element, from a particular Compilation_Unit, from a particular Context. Ids can be written to files. Ids can be read from files and converted into an Element value with the use of a suitable open Context.

21.1 type Id

The Id type represents permanent "names" that correspond to specific Element values.

These names can be written to files, retrieved at a later time, and converted to Element values.

ASIS Ids are a means of identifying particular Element values obtained from a particular physical compilation unit. Ids are "relative names". Each Id value is valid (is usable, makes sense, can be interpreted) only in the context of an appropriate open ASIS Context.

```
type Id is private;
Nil_Id : constant Id;

function "=" (Left  : in Id;
              Right : in Id)
              return Boolean is abstract;
```

Nil_Id is the value of a Id that represents no element.

21.2 function Hash (id)

```
function Hash (The_Id : in Id)
              return Asis.ASIS_Integer;
```

Returns an implementation-defined value which is a function of the value of The_Id. If A and B are ids such that Is_Equal (A, B) is true, Hash(A) equals Hash(B).

21.3 function "<"

```
function "<" (Left  : in Id;
              Right : in Id) return Boolean;
```

Defines a total ordering on Ids.

21.4 function ">"

```
function ">" (Left  : in Id;
              Right : in Id) return Boolean;
```

Equivalent to Right < Left.

21.5 function Is_Nil (id)

```
function Is_Nil (Right : in Id) return Boolean;
```

Right specifies the Id to check.

Returns True if the Id is the Nil_Id, and returns False otherwise.

21.6 function Is_Equal (ids)

```
function Is_Equal (Left  : in Id;
                   Right : in Id) return Boolean;
```

Left specifies the left Id to compare. Right specifies the right Id to compare.

Returns True if Left and Right represent the same physical Id from the same physical compilation unit, and returns False otherwise. The two Ids convert to Is_Identical Elements when converted with the same open ASIS Context.

21.7 function Create_Id

```
function Create_Id (Element : in Asis.Element) return Id;
```

Element specifies any Element value whose Id is desired.

Returns a unique Id value corresponding to this Element, from the corresponding Enclosing_Compilation_Unit and the corresponding Enclosing_Context. The Id value will not be equal ("=") to the Id value for any other Element value unless the two Elements are Is_Identical.

Nil_Id is returned for a Nil_Element.

Element expects any kind of element.

21.8 function Create_Element

```
function Create_Element (The_Id      : in Id;
                         The_Context : in Asis.Context)
                        return Asis.Element;
```

The_Id specifies the Id to be converted to an Element. The_Context specifies the Context containing the Element with this Id.

Returns the Element value corresponding to The_Id. The_Id shall correspond to an Element available from a Compilation_Unit contained by (can be referenced through) The_Context.

Is_Identical (Create_Element (Create_Id(E), Enclosing_Context(Enclosing_Compilation_Unit(E))), E) is True for any element E.

If Create_Element is called twice with the same Id and context, the two results are Is_Identical.

Raises ASIS_Inappropriate_Element if the Element value is not available through The_Context. The Status is Value_Error and the Diagnosis string will attempt to indicate the reason for the failure. (e.g., "Unit is inconsistent", "No such unit", "Element is inconsistent (Unit inconsistent)", etc.)

21.9 function Debug_Image (id)

```
function Debug_Image (The_Id : in Id) return Wide_String;
```

The_Id specifies an Id to convert.

Returns a string value containing implementation-defined debug information associated with the Id including for Nil_Id.

The return value uses Asis.Text.Delimiter_Image to separate the lines of multi-line results. The return value does not end with Asis.Text.Delimiter_Image.

These values are intended for two purposes. They are suitable for inclusion in problem reports sent to the ASIS implementor. They can be presumed to contain information useful when debugging the implementation itself. They are also suitable for use by the ASIS application when printing simple application debugging messages during application development. They are intended to be, to some worthwhile degree, intelligible to the user.

Section 22: package Asis.Data_Decomposition (optional)

Support for data decomposition is optional. The library package Asis.Data_Decomposition shall exist for an implementation that supports data decomposition. If it exists, the package shall provide interfaces equivalent to those described in the following subclauses.

The operations provided by this package may be used to determine the layout of a record or array type in cases where this information is known statically.

Assumptions and Limitations of this Interface:

a) Records, arrays, and their components may be packed.

b) Records, array components, enumerations, and scalar types may have aspect clauses applied to them. This includes scalar types used as record discriminants and array indices.

c) This specification supports two of the three type models discussed below. Models 1 and 2 are supported. Model 3 is not supported.

   1) Simple "static" types contain no variants, have a single fixed 'Size, and all components and attributes are themselves static and/or fully constrained. The size and position for any component of the type can be determined without regard to constraints. For example:

```
type Static_Record is
    record
        F1, F2 : Natural;
        C1     : Wide_Character;
        A1     : Wide_String (1..5);
    end record;

type Static_Discriminated (X : Boolean) is
    record
        F1, F2 : Natural;
        C1     : Wide_Character;
    end record;

type Static_Array   is array (Integer range 1 .. 100) of Boolean;
type Static_Enum    is (One, Two, Three);
type Static_Integer is range 1 .. 512;
type Static_Float   is digits 15 range -100.0 .. 100.0;
type Static_Fixed   is delta 0.1 range -100.0 .. 100.0;
```

   2) Simple "dynamic" types contain one or more components or attributes whose size, position, or value depends on the value of one or more constraints computed at execution time. This means that the size, position, or number of components within the data type cannot be determined without reference to constraint values.

      Records containing components, whose size depends on discriminants of the record, can be handled because the discriminants for a record value are fully specified by the data stream form of the record value. For example:

```
type Dynamic_Length (Length : Natural) is
    record
        S1 : Wide_String (1 .. Length);
    end record;

type Dynamic_Variant (When : Boolean) is
    record
        case When is
            when True =>
                C1 : Wide_Character;
            when False =>
                null;
        end case;
    end record;
```

Arrays with an unconstrained subtype, whose 'Length, 'First, and 'Last depend on dynamic index constraints, can be handled because these attributes can be queried and stored when the data stream is written. For example:

```
I : Integer := Some_Function;
type Dynamic_Array is
    array (Integer range I .. I + 10) of Boolean;

type Heap_Array   is array (Integer range <>) of Boolean;
type Access_Array is access Heap_Array;
X : Access_Array := new Heap_Array (1 .. 100);
```

3) Complex, externally "discriminated" records, contain one or more components whose size or position depends on the value of one or more non-static external values (values not stored within instances of the type) at execution time. The size for a value of the type cannot be determined without reference to these external values, whose runtime values are not known to the ASIS Context and cannot be automatically recorded by the Asis.Data_Decomposition.Portable_Transfer generics. For example: Asis.Data_Decomposition.Portable_Transfer generics. A class-wide type also falls in this category, as does an array type with a dynamic component size. For example:

```
N : Natural := Function_Call();
....
declare
    type Complex is
        record
            S1 : Wide_String (1 .. N);
        end record;
begin
    ....
end;
```

General Usage Rules:

All implementations will handle arrays with a minimum of 16 dimensions, or the number of dimensions allowed by their compiler, whichever is smaller.

22.1 type Record_Component

Type Record_Component describes one discriminant or component of a record type.

The "=" operator is not meaningful between Record_Component values unless one of them is the Nil_Record_Component value.

A record type describes composite values which contain zero or more discriminant and component fields. A_Record_Type_Definition can be queried to obtain a list of Record_Components. Each Record_Component contains the information necessary to extract one discriminant or component field of the record.

Record_Components are intended for use with data stream extraction operations. An extraction operation is performed using a Record_Component, in conjunction with a data stream representing a value of the record type. The record data stream contains data for all fields of the record. The result is an extracted data stream representing just the value of the one field. Record_Components are implemented so as to allow for efficient extraction of field values.

An extracted field data stream is suitable for all uses. If the field is a scalar type, it can be converted directly into useful information. If the field is, in turn, another composite value, it can be further decomposed into its own component values.

There are two ways to obtain the Record_Components or the Array_Component needed to further decompose an embedded composite field. First, if the type of the field is known, the type definition can be directly queried to obtain the Record_Components or the Array_Component that describe its internal structure. Second, the Record_Component used to extract the field can be queried to obtain the same Record_Components or the same Array_Component. Both methods return identical information.

This kind of nested decomposition can be carried to any required level.

Record_Components become invalid when the Context, from which they originate, is closed. All Record_Components are obtained by referencing a) an Element, which has an associated Context, b) another Record_Component, or c) an Array_Component. Ultimately, all component values originate from a A_Type_Definition Element; that Element determines their Context of origin.

```
type Record_Component is private;
Nil_Record_Component : constant Record_Component;

function "=" (Left  : in Record_Component;
              Right : in Record_Component)
              return Boolean is abstract;
```

Nil_Record_Component is the value of a Record_Component that represents no component.

22.2 type Record_Component_List

```
type Record_Component_List is
   array (Asis.List_Index range <>) of Record_Component;
```

Type Record_Component_List represents a list of record components.

22.3 type Array_Component

Type Array_Component describes the components of an array valued field for a record type.

The "=" operator is not meaningful between Array_Component values unless one of them is the Nil_Array_Component value.

An array type describes composite values which contain zero or more indexed components. Both An_Unconstrained_Array_Definition or A_Constrained_Array_Definition can be queried to obtain a single Array_Component. The Array_Component contains the information necessary to extract any arbitrary component of the array.

Array_Components are intended for use with data stream extraction operations. An extraction operation is performed using an Array_Component, in conjunction with a data stream representing a value of the array type. The array data stream contains data for all components of the array. The result is an extracted data stream representing just the value of the one component. Array_Components are implemented so as to allow for efficient extraction of array component values.

An extracted component data stream is suitable for all uses. If the component is a scalar type, it can be converted directly into useful information. If the component is, in turn, another composite value, it can be further decomposed into its own component values.

There are two ways to obtain the Record_Components or the Array_Component needed to further decompose an embedded composite component. First, if the type of the component is known, the type definition can be directly queried to obtain the Record_Components or the Array_Component that describe its internal structure. Second, the Array_Component used to extract the component can be queried to obtain the same Record_Components or the same Array_Component. Both methods return identical information.

This kind of nested decomposition can be carried to any required level.

Array_Components become invalid when the Context, from which they originate, is closed. All Record_Components are obtained by referencing a) an Element, which has an associated Context, b) a Record_Component, or c) another Array_Component. Ultimately, all component

values originate from a A_Type_Definition Element; that Element determines their Context of origin.

```
type Array_Component is private;
Nil_Array_Component : constant Array_Component;

function "=" (Left  : in Array_Component;
              Right : in Array_Component)
              return Boolean is abstract;
```

Nil_Array_Component is the value of an Array_Component that represents no component.

22.4 type Array_Component_List

```
type Array_Component_List is
   array (Asis.List_Index range <>) of Array_Component;
```

Type Array_Component_List represents a list of array components.

22.5 type Dimension_Indexes

Type Dimension_Indexes is an array of index values used to access an array stream.

```
type Dimension_Indexes is
   array (Asis.ASIS_Positive range <>) of Asis.ASIS_Positive;
```

22.6 type Array_Component_Iterator

Type Array_Component_Iterator is used to iterate over successive components of an array.

Iterators can be copied. The copies operate independently (have separate state).

```
type Array_Component_Iterator is private;

Nil_Array_Component_Iterator : constant Array_Component_Iterator;
```

Nil_Array_Component_Iterator is the value of an Array_Component_Iterator that represents no iterator.

22.7 type Type_Model_Kinds

Each Type_Definition fits into one of three type models.

```
type Type_Model_Kinds is (A_Simple_Static_Model,
                          A_Simple_Dynamic_Model,
                          A_Complex_Dynamic_Model,
                          Not_A_Type_Model);          -- Nil arguments
```

22.8 function Type_Model_Kind

```
function Type_Model_Kind (Type_Definition : in Asis.Type_Definition)
                         return Type_Model_Kinds;

function Type_Model_Kind (Component : in Record_Component)
                         return Type_Model_Kinds;

function Type_Model_Kind (Component : in Array_Component)
                         return Type_Model_Kinds;
```

Type_Definition specifies the type definition to query. Component specifies a record field with a record or array type.

Returns the model that best describes the type indicated by the argument. Returns Not_A_Type_Model for any unexpected argument such as a Nil value.

Type_Definition expects an element that has the following Element_Kinds:

A_Type_Definition

Raises ASIS_Inappropriate_Element with a Status of Value_Error for any element that does not have one of these expected kinds.

22.9 function Is_Nil (component)

```
function Is_Nil (Right : in Record_Component) return Boolean;
function Is_Nil (Right : in Array_Component)  return Boolean;
```

Right specifies the component to check.

Returns True if Right is a Nil (or uninitialized) component value.

Returns False for all other values.

Right expects any kind of component.

22.10 function Is_Equal (component)

```
function Is_Equal (Left  : in Record_Component;
                   Right : in Record_Component) return Boolean;
function Is_Equal (Left  : in Array_Component;
                   Right : in Array_Component)  return Boolean;
```

Left specifies the left component to compare. Right specifies the right component to compare.

Returns True if Left and Right represent the same physical component of the same record or array type from the same physical compilation unit, and returns False otherwise. The two components may or may not be from the same open ASIS Context variable.

Implies:

```
Is_Equal (Enclosing_Compilation_Unit (Component_Declaration (Left)),
          Enclosing_Compilation_Unit (Component_Declaration (Right))) =
True
```

Right and Left expect any kind of component.

22.11 function Is_Identical (component)

```
function Is_Identical (Left  : in Record_Component;
                       Right : in Record_Component) return Boolean;
function Is_Identical (Left  : in Array_Component;
                       Right : in Array_Component)  return Boolean;
```

Left specifies the left component to compare. Right specifies the right component to compare.

Returns True if Left and Right represent the same physical component of the same record or array type from the same physical compilation unit and the same open ASIS Context variable, and returns False otherwise.

Implies:

```
Is_Identical (Enclosing_Compilation_Unit (Component_Declaration (Left)),
              Enclosing_Compilation_Unit (Component_Declaration (Right)))
= True
```

Right and Left expect any kind of component.

22.12 function Is_Array

```
function Is_Array (Component : in Record_Component) return Boolean;
function Is_Array (Component : in Array_Component)  return Boolean;
```

Component specifies any component.

Returns True if the component has an array subtype (contains an array value).

Returns False for Nil components and any component that is not an embedded array.

22.13 function Is_Record

```
function Is_Record (Component : in Record_Component) return Boolean;
function Is_Record (Component : in Array_Component)  return Boolean;
```

Component specifies any component.

Returns True if the component has a record, task, or protected subtype. Returns True for a task or protected component because such a component may have discriminants. Returns False otherwise.

22.14 function Done

```
function Done (Iterator : in Array_Component_Iterator) return Boolean;
```

Iterator specifies the iterator to query.

Returns True if the iterator has been advanced past the last array component. Returns True for a Nil_Array_Component_Iterator. Returns False otherwise.

22.15 procedure Next

```
procedure Next (Iterator : in out Array_Component_Iterator);
```

Iterator specifies the iterator to advance.

Advances the iterator to the next array component. Use Done to test the iterator to see if it has passed the last component. Does nothing if the iterator is Nil_Array_Component_Iterator or is already past the last component.

22.16 procedure Reset

```
procedure Reset (Iterator : in out Array_Component_Iterator);
```

Iterator specifies the iterator to reset.

Resets the iterator to the first array component. Has no effect if Iterator is Nil_Array_Component_Iterator.

22.17 function Array_Index

```
function Array_Index (Iterator : in Array_Component_Iterator)
                     return Asis.ASIS_Natural;
```

Iterator specifies the iterator to query.

Returns the Index value which, when used in conjunction with the Array_Component value used to create the Iterator, indexes the same array component as that presently addressed by the Iterator.

Raises ASIS_Inappropriate_Element if given a Nil_Array_Component_Iterator or one where Done(Iterator) = True. The Status value is Data_Error. The Diagnosis string will indicate the kind of error detected.

22.18 function Array_Indexes

```
function Array_Indexes (Iterator : in Array_Component_Iterator)
                        return Dimension_Indexes;
```

Iterator specifies the iterator to query.

Returns the index values which, when used in conjunction with the Array_Component value used to create the Iterator, indexes the same array component as that presently addressed by the Iterator.

Raises ASIS_Inappropriate_Element if given a Nil_Array_Component_Iterator or one where Done(Iterator) = True. The Status value is Data_Error. The Diagnosis string will indicate the kind of error detected.

22.19 function Discriminant_Components

```
function Discriminant_Components (Type_Definition : in Asis.Type_Definition)
                                  return Record_Component_List;
function Discriminant_Components (Component : in Record_Component)
                                  return Record_Component_List;
function Discriminant_Components (Component : in Array_Component)
                                  return Record_Component_List;
```

Type_Definition specifies the record type definition to query. Component specifies a component which has a record subtype, Is_Record(Component) = True.

Returns a list of the discriminant components for records of the indicated record type.

The result describes the locations of the record type's discriminants, regardless of the static or dynamic nature of the record type.

All return values are valid parameters for all query operations that accept Record_Components.

Type_Definition expects an element that has the following Element_Kinds:

A_Type_Definition that has one of the following Type_Kinds:
A_Derived_Type_Definition (derived from a record type)
A_Record_Type_Definition

Component expects a component that has Is_Record(Component) = True and has one of the following Asis.Data_Decomposition.Type_Model_Kinds:

A_Simple_Static_Model
A_Simple_Dynamic_Model
A_Complex_Dynamic_Model

Raises ASIS_Inappropriate_Element with a Status of Value_Error for any element or component that does not have one of these expected kinds or types.

22.20 function Record_Components (data decomposition)

```
function Record_Components (Type_Definition : in Asis.Type_Definition)
                           return Record_Component_List;
function Record_Components (Component : in Record_Component)
                           return Record_Component_List;
function Record_Components (Component : in Array_Component)
                           return Record_Component_List;
```

Type_Definition specifies the record type definition to query. Component specifies a component which has a record subtype, Is_Record(Component) = True.

Returns a list of the discriminants and components for the indicated simple static record type.

The result describes the locations of the record type's discriminants and components.

All return values are valid parameters for all query operations that accept Record_Components.

NOTE If an Ada implementation uses implementation-dependent record components (Ada Standard 13.5.1 (15)), then each such component of the record type is included in the result.

Type_Definition expects an element that has the following Element_Kinds:

A_Type_Definition that has one of Type_Kinds:
A_Derived_Type_Definition (derived from a record type)
A_Record_Type_Definition

Component expects a component that has Is_Record(Component) = True and has the following Asis.Data_Decomposition.Type_Model_Kinds:

A_Simple_Static_Model

Raises ASIS_Inappropriate_Element with a Status of Value_Error for any element or component that does not have one of these expected kinds or types.

22.21 function Array_Components

```
function Array_Components (Type_Definition : in Asis.Type_Definition)
                          return Array_Component;
function Array_Components (Component : in Record_Component)
                          return Array_Component;
function Array_Components (Component : in Array_Component)
                          return Array_Component;
```

Type_Definition specifies the array type definition to query. Component specifies a component which has an array subtype, Is_Array(Component) = True.

Returns a single component, describing all components of the indicated array type. The array type shall be a simple static, or a simple dynamic array type.

The result contains all information necessary to index and extract any component of a data stream representing a value of the indicated array type.

All return values are valid parameters for all query operations that accept Array_Components.

Type_Definition expects an element that has the following Element_Kinds:

A_Type_Definition that has one of the following Type_Kinds:
A_Derived_Type_Definition (derived from an array type)
An_Unconstrained_Array_Definition
A_Constrained_Array_Definition

Component expects a component that has Is_Array(Component) = True and has one of the following Asis.Data_Decomposition.Type_Model_Kinds:

A_Simple_Static_Model
A_Simple_Dynamic_Model

Raises ASIS_Inappropriate_Element with a Status of Value_Error for any element or component that does not have one of these expected kinds or types.

22.22 function Array_Iterator

```
function Array_Iterator (Type_Definition : in Asis.Type_Definition)
                         return Array_Component_Iterator;
function Array_Iterator (Component : in Record_Component)
                         return Array_Component_Iterator;
function Array_Iterator (Component : in Array_Component)
                         return Array_Component_Iterator;
```

Type_Definition specifies the array type definition to query. Component specifies a component which has an array subtype, Is_Array(Component) = True.

Returns an iterator poised to fetch the 1st component of an array.

Type_Definition expects an element that has the following Element_Kinds:

A_Type_Definition that has one of the following Type_Kinds:
A_Derived_Type_Definition (derived from an array type)
An_Unconstrained_Array_Definition
A_Constrained_Array_Definition

Component expects a component that has Is_Array(Component) = True and has one of the following Asis.Data_Decomposition.Type_Model_Kinds:

A_Simple_Static_Model
A_Simple_Dynamic_Model

Raises ASIS_Inappropriate_Element with a Status of Value_Error for any element or component that does not have one of these expected kinds or types.

22.23 function Component_Declaration

```
function Component_Declaration (Component : in Record_Component)
                                return Asis.Declaration;
```

Component specifies the component to be queried

Returns an Asis.Declaration, which is either A_Component_Declaration or A_Discriminant_Specification. These values can be used to determine the subtype, type, and base type of the record component. The result may be an explicit declaration made by the user, or, it may be an implicit component declaration for an implementation-defined component (Ada Standard 13.5.1(15)).

Component expects any kind of non-Nil component.

Raises ASIS_Inappropriate_Element with a Status of Value_Error for any element or component that does not have one of these expected kinds.

Returns an element that has the following Element_Kinds:

A_Declaration that has one of the following Declaration_Kinds:
A_Component_Declaration
A_Discriminant_Specification

22.24 function Component_Indication

```
function Component_Indication (Component : in Array_Component)
                               return Asis.Subtype_Indication;
```

Component specifies the component to be queried.

Returns an Asis.Subtype_Indication. These values can be used to determine the subtype, type, and base type of the array components.

Component expects any kind of non-Nil component.

Raises ASIS_Inappropriate_Element with a Status of Value_Error for any element or component that does not have one of these expected kinds.

Returns an element that has the following Element_Kinds:

A_Subtype_Indication

22.25 function All_Named_Components

```
function All_Named_Components (Type_Definition : in Asis.Type_Definition)
                               return Asis.Defining_Name_List;
```

Type_Definition specifies the record type definition to query.

Returns a list of all discriminant and component entity names defined by the record type. All record type definitions are appropriate for this operation. This query provides a means for determining whether a field, with a particular name, exists for some possible instance of the record type. This list does not include the names of implementation-defined components (Ada Standard 13.5.1 (15)); those name have the form of An_Attribute_Reference expression.

Type_Definition expects an element that has the following Element_Kinds:

A_Type_Definition that has one of the following Type_Kinds:
A_Derived_Type_Definition (derived from a record type)
A_Record_Type_Definition

and Type_Definition expects an element that has one of the following Asis.Data_Decomposition.Type_Model_Kinds:

A_Simple_Static_Model
A_Simple_Dynamic_Model
A_Complex_Dynamic_Model

Raises ASIS_Inappropriate_Element with a Status of Value_Error for any element that does not have one of these expected kinds.

22.26 function Array_Length

```
function Array_Length (Component : in Record_Component)
                       return Asis.ASIS_Natural;
function Array_Length (Component : in Array_Component)
                       return Asis.ASIS_Natural;
```

Component specifies the component to query.

Returns the number of components within an array valued component. The array subtype may be multidimensional. The result treats the array as if it were unidimensional. It is the product of the 'Lengths of the individual array dimensions.

Component expects a component that has Is_Array(Component) = True.

Raises ASIS_Inappropriate_Element with a Status of Value_Error for any component that is not an array component.

22.27 function Array_Length (with dimension)

```
function Array_Length (Component : in Record_Component;
                       Dimension : in Asis.ASIS_Natural)
                      return Asis.ASIS_Natural;

function Array_Length (Component : in Array_Component;
                       Dimension : in Asis.ASIS_Natural)
                       return Asis.ASIS_Natural;
```

Component specifies the component to query. Dimension specifies the array dimension to query.

Returns the number of components within an array valued component. The array subtype may be unidimensional. The result is the 'Length(Dimension) of the array.

Component expects a component that has Is_Array(Component) = True.

Raises ASIS_Inappropriate_Element with a Status of Value_Error for any component that is not an array component.

22.28 function Size

```
function Size (Type_Definition : in Asis.Type_Definition)
              return Asis.ASIS_Natural;

function Size (Component : in Record_Component) return Asis.ASIS_Natural;

function Size (Component : in Array_Component)  return Asis.ASIS_Natural;
```

Type_Definition specifies a type definition, whose 'Size is desired. Component specifies a component, whose 'Size is desired.

Returns the minimum number of bits required to hold a simple static type, the number of bits allocated to hold a record field, or the number of bits allocated to hold each array component.

Type_Definition expects an element of Element_Kinds:

A_Type_Definition

Component expects a component that has the following Asis.Data_Decomposition.Type_Model_Kinds:

A_Simple_Static_Model

Raises ASIS_Inappropriate_Element with a Status of Value_Error for any element or component that does not have one of these expected kinds.

22.29 function Position

```
function Position (Component : in Record_Component)
                  return Asis.ASIS_Natural;

function Position (Component : in Array_Component;
                   Index     : in Asis.ASIS_Positive)
                  return Asis.ASIS_Natural;

function Position (Component : in Array_Component;
                   Indexes   : in Dimension_Indexes)
                  return Asis.ASIS_Natural;

function Position (Iterator : in Array_Component_Iterator)
                  return Asis.ASIS_Natural;
```

Component specifies the component to query. Index specifies a value in the range 1..Array_Length (Component), the index of the component to query. Indexes specifies a list of index values, there is one value for each dimension of the array type, each index N is in the range 1..Array_Length (Component, N);. Iterator specifies a particular array component to query.

Returns the System.Storage_Unit offset, from the start of the first storage unit occupied by the enclosing composite type, of the first of the storage units occupied by the Component. The offset is measured in storage units.

Component expects any kind of non-Nil component. Raises ASIS_Inappropriate_Element with a Status of Value_Error for any Nil component. Raises ASIS_Inappropriate_Element with a Status of Data_Error if any index is not in the expected range or if Done (Iterator) = True. The Status value will be Data_Error. The Diagnosis string will indicate the kind of error detected.

22.30 function First_Bit

```
function First_Bit (Component : in Record_Component)
                   return Asis.ASIS_Natural;

function First_Bit (Component : in Array_Component;
                    Index     : in Asis.ASIS_Positive)
                   return Asis.ASIS_Natural;

function First_Bit (Component : in Array_Component;
                    Indexes   : in Dimension_Indexes)
                   return Asis.ASIS_Natural;

function First_Bit (Iterator : in Array_Component_Iterator)
                   return Asis.ASIS_Natural;
```

Component specifies the component to query. Index specifies a value in the range 1..Array_Length (Component), the index of the component to query. Indexes specifies a list of index values, there is one value for each dimension of the array type, each index N is in the range 1..Array_Length (Component, N);. Iterator specifies a particular array component to query.

Returns the bit offset, from the start of the first of the storage units occupied by the Component, of the first bit occupied by the Component. The offset is measured in bits.

Component expects any kind of non-Nil component. Raises ASIS_Inappropriate_Element with a Status of Value_Error for any Nil component. Raises ASIS_Inappropriate_Element with a Status of Data_Error if any index is not in the expected range or if Done (Iterator) = True. The Status value will be Data_Error. The Diagnosis string will indicate the kind of error detected.

22.31 function Last_Bit

```
function Last_Bit (Component : in Record_Component)
                  return Asis.ASIS_Natural;

function Last_Bit (Component : in Array_Component;
                   Index     : in Asis.ASIS_Positive)
                  return Asis.ASIS_Natural;

function Last_Bit (Component : in Array_Component;
                   Indexes   : in Dimension_Indexes)
                  return Asis.ASIS_Natural;

function Last_Bit (Iterator : in Array_Component_Iterator)
                  return Asis.ASIS_Natural;
```

Component specifies the component to query. Index specifies a value in the range 1..Array_Length (Component), the index of the component to query. Indexes specifies a

list of index values, there is one value for each dimension of the array type, each index N is in the range 1..Array_Length (Component, N);. Iterator specifies a particular array component to query.

Returns the bit offset, from the start of the first of the storage units occupied by the Index'th Element, of the last bit occupied by the Element. The offset is measured in bits.

Component expects any kind of non-Nil component. Raises ASIS_Inappropriate_Element with a Status of Value_Error for any Nil component. Raises ASIS_Inappropriate_Element with a Status of Data_Error if any index is not in the expected range or if Done (Iterator) = True. The Status value will be Data_Error. The Diagnosis string will indicate the kind of error detected.

Section 23: ASIS Semantic Subsystem

23.1 Introduction for ASIS Semantic Subsystem

The ASIS Semantic Subsystem comprises a set of packages that provide a semantic-level interface to the program library. Other parts of ASIS are based on Elements that represent syntactic constructs. The Semantic Subsystem defines a set of types that represent views of entities defined by syntactic constructs. The type View is the most general, representing a view of essentially any kind of program entity. There are seven first-level extensions of View defined: Subtype_View, Object_View, Callable_View, Package_View, Generic_View, Statement_View, and Exception_View. Elementary_Subtype and Composite_Subtype are extensions of Subtype_View, and there are further extensions for particular subcategories of types.

Object_View is used to represent a view that might denote either an object or a pure value. Callable_View is used to represent subprograms, entries, accept statements, etc. A Package_View is used to represent either the full view or the limited view of a package. Generic_View, Statement_View, and Exception_View are used to represent generic units, statements, and exceptions, respectively.

In addition to View and its various extensions, the Semantic Subsystem includes a type View_Declaration used to represent the declaration of a View. These are grouped into Region_Parts, which in turn are grouped into Declarative_Regions.

There are various operations for navigating back and forth between the Semantic Subsystem and other parts of ASIS. When starting from the Semantic Subsystem, given a View_Declaration, there is an operation that identifies the corresponding Asis.Declaration and one that identifies the associated Asis.Identifier. Given a View, there is an operation that identifies the Asis.Expression that denotes it, if any. Given a Declarative_Region, there is an operation that identifies the Asis.Element that defines it, if any.

When starting from an Asis.Element, there is an operation that given an Asis.Declaration can identify the corresponding View_Declaration, one that given an Asis.Type_Definition can identify the corresponding Subtype_View, and one that given an Asis.Expression can identify the corresponding Object_View (or more generally "View" in the case where the Expression is a Name).

The generic packages Containers.Vectors and Containers.Indefinite_Holders are instantiated as needed for various types used in the ASIS Semantic Subsystem. These instantiations permit more convenient manipulation of objects of a class-wide type, allowing for extensible lists and more easily updatable variables, despite the fact that class-wide types are indefinite, and normally require initialization at the point of declaration.

NOTE 1 The semantic subsystem does not include a value that represents no view. Such a value would be difficult to define for an interface type (which allows no objects), and it is not needed for the intended use of the views. A holder container in the empty state can mean "no view" if the program requires that functionality.

NOTE 2 Exception ASIS_Not_In_Context is raised for any operation defined in the packages of the semantic subsystem, if the result is an entity that is not part of the current context (see Section 5). An entity could not be present in the current context because, for instance, if the View is of an incomplete view, and the full view is not included in the context. It also can happen if only a portion of the semantic dependencies of the queried unit is included in the context, and the query returns an entity defined in one of the semantic dependencies that is not included in the context.

23.2 package Asis.Views

The library package Asis.Views shall exist. The package shall provide interfaces equivalent to those described in the following subclauses.

23.2.1 type View_Kinds and type View

```
type View_Kinds is (
   An_Exception,
   An_Exception_Renaming,

   A_Generic_Package,
   A_Generic_Package_Renaming,
   A_Generic_Subprogram,
   A_Generic_Subprogram_Renaming,

   A_Noninstance_Package,
   A_Package_Instance,
   A_Package_Renaming,
   A_Limited_Package_View,

   A_Noninstance_Subprogram,
   A_Subprogram_Instance,
   A_Subprogram_Renaming,
   A_Protected_Subprogram,
   An_Imported_Subprogram,
   An_Attribute_Subprogram,
   An_Intrinsic_Subprogram,
   A_Designated_Subprogram,
   A_Generic_Formal_Subprogram,
   A_Protected_Entry,
   A_Task_Entry,

   A_Standalone_Object,
   A_Generic_Formal_Object,
   A_Formal_Parameter_Object,
   An_Object_Renaming,
   A_Designated_Object,
   A_Component_Object,
   An_Attribute_Object,
   An_Aggregate_Object,
   A_Function_Result_Object,
   A_Named_Number,
   An_Attribute_Value,
   An_Expression_Value,

   A_Boolean_Subtype,
   A_Character_Subtype,
   An_Ordinary_Enumeration_Subtype,
   A_Signed_Integer_Subtype,
   A_Modular_Integer_Subtype,
   An_Ordinary_Fixed_Subtype,
   A_Decimal_Fixed_Subtype,
   A_Float_Subtype,
   An_Access_To_Object_Subtype,
   An_Access_To_Subprogram_Subtype,

   An_Array_Subtype,
   A_Record_Subtype,
   A_Record_Extension,
   A_Task_Subtype,
   A_Protected_Subtype,
   An_Interface,

   A_Private_Subtype,
   A_Private_Extension,

   An_Incomplete_Subtype,

   A_Label_Statement_View,
   A_Block_Statement_View,
   A_Loop_Statement_View);

subtype Exception_View_Kinds is View_Kinds
   range An_Exception .. An_Exception_Renaming;

subtype Generic_View_Kinds is View_Kinds
   range A_Generic_Package .. A_Generic_Subprogram_Renaming;

subtype Callable_View_Kinds is View_Kinds
   range A_Noninstance_Subprogram .. A_Task_Entry;
```

```
subtype Object_View_Kinds is View_Kinds
   range A_Standalone_Object .. An_Expression_Value;
subtype Subtype_View_Kinds is View_Kinds
   range A_Boolean_Subtype .. An_Incomplete_Subtype;
subtype Package_View_Kinds is View_Kinds range
   A_Noninstance_Package..A_Limited_Package_View;
subtype Statement_View_Kinds is View_Kinds range
   A_Label_Statement_View .. A_Loop_Statement_View;
type View is interface;
```

The tagged type View is the root of a hierarchy of types used to represent views on program entities. The distinct kinds of entities are enumerated by the type View_Kind. Subranges of this enumeration type are defined to correspond to the extensions of the View type.

```
function Kind (V : View) return View_Kinds is abstract;
function Is_Object_Or_Value (V : View) return Boolean is abstract;
function Is_Callable (V : View) return Boolean is abstract;
function Is_Subtype (V : View) return Boolean is abstract;
function Is_Package (V : View) return Boolean is abstract;
function Is_Generic (V : View) return Boolean is abstract;
function Is_Exception (V : View) return Boolean is abstract;
function Is_Statement (V : View) return Boolean is abstract;
```

V specifies the view to query for each of these functions.

The View_Kind of an object V whose type is covered by View'Class may be determined by the Kind dispatching operation. In addition, Boolean queries Is_Object_Or_View, Is_Callable, Is_Subtype, Is_Package, Is_Generic, Is_Exception, and Is_Statement, are provided to determine to which extension of View the specified view V belongs.

23.2.2 function Element_Denoting_View

```
function Element_Denoting_View (V : View) return Asis.Element is abstract;
```

V specifies the view to query.

Returns an Asis.Element that denotes the View V. This is one of the primary mechanisms for navigating from the Semantic Subsystem to the other parts of ASIS.

NOTE This is the element that includes the syntactic construct that caused the particular usage represented by the view. This is not the declaration of the view unless the view is of a declaration.

Examples

For instance, in the following program fragment:

```
declare
   Var : Boolean := Func; -- (1)
begin
   loop
      exit when Var; -- (2)
      ...
```

If V is a view representing the use of Var in the exit statement at (2), the element returned by Element_Denoting_View is the one in the exit statement, and not the one in the declaration. In particular, if E is the element representing the exit statement, then Exit_Condition (E) = Element_Denoting_View (V).

23.2.3 type Conventions

```
type Conventions is (
   Intrinsic_Convention,
   Ada_Convention,
   Protected_Convention,
   Entry_Convention,
   Other_Convention,
   Unspecified_Convention);

function Convention (V : View) return Conventions is abstract;

function Convention_Identifier (V : View) return Wide_String is abstract;
```

V specifies the view to query for both of these functions.

Each program entity can have an associated convention. The convention of the entity identified by the view V is given by the functions Convention and Convention_Identifier. Function Convention_Identifier returns "Intrinsic", "Ada", "Protected", or "Entry" when function Convention returns the corresponding value of type Conventions. When Convention returns Other_Convention, Convention_Identifier returns the convention identifier given in the pragma Convention, Import, or Export used to specify the convention of the entity.

23.2.4 subpackage Declarative_Regions

The subpackage Declarative_Regions within package Views provides semantic information about the declaration, if any, associated with a given view of an entity. This includes the identifier introduced by the declaration to denote the view, as well as indicating the particular part of the declarative region in which the declaration occurred. The subpackage has the contents given in the following subclauses.

23.2.5 type Declarative_Region and type View_Declaration

```
type Declarative_Region is private;

type View_Declaration is interface;
```

A View_Declaration specifies the declaration, if any, that defines a given view. A Declarative_Region represents a declarative region in which declarations may occur.

```
function Defined_View (D : View_Declaration) return View'Class is abstract;

function Declaration (D : View_Declaration)
   return Asis.Declaration is abstract;

function View_Defining_Name (D : View_Declaration)
   return Asis.Defining_Name is abstract;

function Is_Imported (D : View_Declaration) return Boolean is abstract;

function Enclosing_Region (D : View_Declaration)
   return Declarative_Region is abstract;
```

D specifies the view declaration to query for each of these functions.

Function Defined_View returns the View defined by declaration D. If a declaration defines multiple views because the list of Identifiers has more that one element, then a separate View_Declaration is associated with each view, just as though they were defined by separate declarations. Function Declaration returns the Asis.Declaration corresponding to the declaration D.

Function View_Defining_Name returns the Asis.Defining_Name introduced by the declaration D. Nil_Element is returned if D is anonymous (does not introduce a name). For most declarations, View_Defining_Name returns an identifier, but it can also return

an operator symbol, a character literal (for an enumeration value), or an expanded name (for a child unit).

Function Is_Imported returns True if and only if the declaration D is completed with a pragma Import.

Function Enclosing_Region returns the Declarative_Region enclosing the declaration D.

```
function Are_Declared_In_Same_Region (Decl1, Decl2 : View_Declaration)
   return Boolean is abstract;
```

Decl1 and Decl2 specify the view declarations to compare.

Returns True if Enclosing_Region (Decl1) and Enclosing_Region (Decl2) represent the same region, and returns False otherwise.

```
function Is_Declared_Earlier_In_Same_Region (Earlier, Later : View_Declaration)
   return Boolean is abstract;
```

Earlier and Later specify the view declarations to compare.

Returns True if Are_Declared_In_Same_Region (Earlier, Later) is True and Earlier occurs before Later in the logical sequence of the program, where a package specification is presumed to occur before the corresponding package body in this sequence. Otherwise returns False.

23.2.6 type Region_Part and type Region_Part_Kinds

```
type Region_Part is private;

type Region_Part_List is array (Positive range <>) of Region_Part;

type Region_Part_Kinds is (
   Generic_Formal_Part,
   Callable_Formal_Part,
   Discriminant_Part,
   Entry_Family_Index_Part,
   Record_Part,
   Extension_Part,
   Package_Visible_Part,
   Package_Private_Part,
   Task_Visible_Part,
   Task_Private_Part,
   Protected_Visible_Part,
   Protected_Private_Part,
   Body_Part,
   Block_Declarative_Part,
   Loop_Declarative_Part,
   Child_Part);
```

Certain declarative regions are broken up into distinct parts, represented by the type Region_Part. The type Region_Part_Kinds enumerates the distinct kinds of region parts.

```
function Kind (P : Region_Part) return Region_Part_Kinds;

function Is_Empty (P : Region_Part) return Boolean;

function Declarative_Items (P : Region_Part)
   return Asis.Declarative_Item_List;

procedure Declarations (P : Region_Part;
   Declarations : out View_Declaration_Vector'Class);

function Region (P : Region_Part) return Declarative_Region;
```

P specifies the region part to query for each of these subprograms.

Function Kind returns the kind of the region part P. Function Is_Empty returns True if and only if the region part P has no declarative items within it. Function Declarative_Items returns the list of Asis.Declarative_Items that occur within the region part P. Procedure Declarations returns in the parameter Declarations a vector comprising the declarations

that occur within the region part P. Function Region returns the Declarative_Region of which the region part P is a part.

```
function All_Region_Parts (R : Declarative_Region)
   return Region_Part_List;

function Visible_Region_Parts (R : Declarative_Region)
   return Region_Part_List;

function Private_Part (R : Declarative_Region)
   return Region_Part;

function Body_Part (R : Declarative_Region)
   return Region_Part;
```

R specifies the declarative region to query for each of these functions.

Function All_Region_Parts returns an array of the region parts comprising the declarative region R. Function Visible_Region_Parts returns an array of the visible parts of the declarative region R.

Function Private_Part returns the private part of the declarative region R. Private_Part returns an empty Region_Part if R has no private part, or if the private part of R is empty. Function Body_Part returns the body part of the declarative region R. Body_Part returns an empty Region_Part if R has no body part, or if the body part of R is empty. The declarative region of a loop statement or of a block statement has only a body part. Other kinds of regions may have multiple region parts.

```
function Enclosing_Region_Part (D : View_Declaration)
   return Region_Part is abstract;
```

D specifies the view declaration to query.

Returns the Region_Part in which the declaration D occurs.

23.2.7 Nested Declarative Regions

```
function Defining_Construct (R : Declarative_Region)
   return Asis.Element;

function Has_Defining_Declaration (R : Declarative_Region) return Boolean;

function Defining_Declaration (R : Declarative_Region)
   return View_Declaration'Class;

function Has_Enclosing_Region (R : Declarative_Region) return Boolean;

function Enclosing_Region (R : Declarative_Region)
   return Declarative_Region;
```

R specifies the declarative region to query for each of these functions.

Declarative regions are generally associated with language constructs, and may be nested. Function Defining_Construct returns the Asis.Element with which the declarative region R is associated.

Function Has_Defining_Declaration returns True if and only if the declarative region R is associated with an enclosing declaration. Function Defining_Declaration returns the declaration that defines the region R. Defining_Declaration raises ASIS_Inappropriate_View if Has_Defining_Declaration (R) returns False.

Function Has_Enclosing_Region returns True if and only the region R has an enclosing region. Function Enclosing_Region returns the enclosing region of region R, or raises ASIS_Inappropriate_View if Has_Enclosing_Region (R) returns False.

```
function Enclosing_Compilation_Unit (D : View_Declaration)
   return Asis.Compilation_Unit is abstract;

function Expanded_Name (D : View_Declaration)
   return Wide_String is abstract;
```

D specifies the view declaration to query for each of these functions.

Function Enclosing_Compilation_Unit returns the Asis.Compilation_Unit representing the compilation unit in which the declaration D occurs. Function Expanded_Name returns the Wide_String representing the full expanded name denoting the declaration D (encoded in UTF-16, as described in 3). The result is implementation defined if D is declared within an unnamed block or loop statement.

23.2.8 Overloading, Overriding, and Renaming

```
function Is_Overloadable (D : View_Declaration)
   return Boolean is abstract;

function Is_Overriding (D : View_Declaration)
   return Boolean is abstract;

procedure Overridden_Declarations (D : View_Declaration;
   Overridden : out View_Declaration_Vector'Class) is abstract;

function Is_Renaming (D : View_Declaration) return Boolean is abstract;

function Renamed_View (D : View_Declaration) return View'Class is abstract;

function Is_Renaming_As_Body (D : View_Declaration) return Boolean is abstract;
```

D specifies the view declaration to query for each of these subprograms.

Declarations may overload, override, or rename views of other declarations. Function Is_Overloadable returns True if and only if the declaration D is a declaration that may be overloaded. Function Is_Overriding returns True if and only if the declaration D overrides one or more other declarations. Procedure Overridden_Declarations returns (in the Overridden parameter) a vector of those declarations overridden by the declaration D. Overridden will be an empty vector if D does not override any other declaration. Function Is_Renaming returns True if and only if the declaration D is a renaming of another view. Function Renamed_View returns the view that the declaration D is a renaming of, or raises ASIS_Inappropriate_View if Is_Renaming (D) returns False. Function Is_Renaming_As_Body returns True if and only if the declaration D is a renaming-as-body.

23.2.9 Aspect Items

```
function Aspect_Items (D : View_Declaration)
    return Asis.Aspect_Clause_List is abstract;
```

D specifies the view declaration to query.

Function Aspect_Items returns a list of aspect clauses that are visible for the view of the declaration D.

23.2.10 View Declaration Vectors

```
package View_Declaration_Vectors is
   new Ada.Containers.Indefinite_Vectors (Positive, View_Declaration'Class);
type View_Declaration_Vector is new
   View_Declaration_Vectors.Vector with null record;
```

Type View_Declaration_Vector allows the declaration of a list of View_Declarations.

These are the last interfaces defined in subpackage Declarative_Regions; following subclauses contain interfaces directly defined in package Asis.Views.

23.2.11 Views and Declarations

```
function Has_Declaration (V : View) return Boolean is abstract;

function Declaration (V : View) return View_Declaration'Class is abstract;
```

```
function Defines_Declarative_Region (V : View) return Boolean is abstract;
function Defined_Region (V : View) return Declarative_Region is abstract;
```

V specifies the view to query for all of these functions.

Function Has_Declaration returns True if and only if view V was defined by a declaration. Function Declaration returns the declaration that defines view V, or raises ASIS_Inappropriate_View if Has_Declaration (V) returns False.

Function Defines_Declarative_Region returns True if and only if view V is of an entity that has its own declarative region. Function Defined_Region returns the declarative region of the entity represented by view V, or raises ASIS_Inappropriate_View if Defines_Declarative_Region (V) returns False.

23.2.12 Representational and Operational Aspects

```
type Language_Defined_Aspect_Kinds is (
    Address,
    Alignment,
    Asynchronous,
    Atomic,
    Atomic_Components,
    Bit_Order,
    Coding,
    Component_Size,
    Controlled,
    Convention,
    Discarded_Names,
    Exported,
    External_Tag,
    Imported,
    Independent,
    Independent_Components,
    Input,
    Layout,
    No_Return,
    Output,
    Packing,
    Read,
    Size,
    Small,
    Storage_Pool,
    Storage_Size,
    Stream_Size,
    Volatile,
    Volatile_Components,
    Write);
function Is_Aspect_Specified (V : View;
   Aspect : Language_Defined_Aspect_Kinds) return Boolean is abstract;
function Is_Aspect_Directly_Specified (V : View;
   Aspect : Language_Defined_Aspect_Kinds) return Boolean is abstract;
```

V specifies the view to query and Aspect specifies the language-defined aspect to query for both of these functions.

Function Is_Aspect_Specified returns True if the representational or operational aspect Aspect is specified for the declaration of the view V, and returns False otherwise. Function Is_Aspect_Directly_Specified returns True if the representational or operational aspect Aspect is directly specified for the declaration of the view V, and returns False otherwise.

These functions return False if the Aspect named cannot be specified for the kind of entity represented by V, or if the view V does not have a declaration.

```
type Implementation_Defined_Aspect_Kinds is (<implementation-defined>);
function Is_Aspect_Specified (V : View;
   Aspect : Implementation_Defined_Aspect_Kinds) return Boolean is abstract;
function Is_Aspect_Directly_Specified (V : View;
   Aspect : Implementation_Defined_Aspect_Kinds) return Boolean is abstract;
```

Type Implementation_Defined_Aspect_Kinds is an implementation-defined enumeration of aspect names; it should include the names of all implementation-defined aspects defined by the implementation. If there are no implementation-defined aspects, it should consist of the single item None.

V specifies the view to query and Aspect specifies the implementation-defined aspect to query for both of these functions.

Function Is_Aspect_Specified returns True if the representational or operational aspect Aspect is specified for the declaration of the view V, and returns False otherwise. Function Is_Aspect_Directly_Specified returns True if the representational or operational aspect Aspect is directly specified for the view V, and returns False otherwise.

These functions return False if the named aspect cannot be specified for the kind of entity represented by V, or if the view V does not have a declaration.

NOTE To find out all of the aspect clauses for an entity, use function Aspect_Items on the declaration of the entity.

23.2.13 View Holders

```
package View_Holders is new Ada.Containers.Indefinite_Holders (View'Class);
type View_Holder is new View_Holders.Holder with null record;
```

Type View_Holder allows the declaration of an uninitialized variable (including a component) that can hold a View.

23.2.14 View Vectors

```
package View_Vectors is new
   Ada.Containers.Indefinite_Vectors (Positive, View'Class);
type View_Vector is new View_Vectors.Vector with null record;
```

Type View_Vector allows the declaration of a list of Views.

23.3 package Asis.Program_Units

The library package Asis.Program_Units shall exist. The package shall provide interfaces equivalent to those described in the following subclauses.

23.3.1 type Program_Unit

```
type Program_Unit is interface and View_Declaration;
```

The type Program_Unit is an extension of View_Declaration, and is used to represent the declaration of a program unit (package, subprogram, task unit, protected unit, protected entry, or generic unit).

```
function Requires_Completion (P : Program_Unit) return Boolean is abstract;
function Has_Body (P : Program_Unit) return Boolean is abstract;
function Body_Of_Program_Unit (P : Program_Unit)
   return Asis.Declaration is abstract;
function Is_Instance (P : Program_Unit) return Boolean is abstract;
function Instantiated_Generic (P : Program_Unit)
   return Program_Unit'Class is abstract;
```

```
function Actual_Part (P : Program_Unit)
   return Asis.Association_List is abstract;

function Expanded_Body (P : Program_Unit)
   return Asis.Declaration is abstract;
```

P specifies the program unit view declaration to query for each of these functions.

Function Requires_Completion returns True if and only if the program unit P requires a completion. Function Has_Body returns True if the program unit P has a completion that is a body. Function Body_Of_Program_Unit returns the body that completes the program unit P, or raises ASIS_Inappropriate_View if Has_Body (P) returns False.

Function Is_Instance returns True if and only if the program unit P is an instance of a generic unit. Function Instantiated_Generic returns the generic unit given the instance P, or raises ASIS_Inappropriate_View if Is_Instance (P) returns False. Function Actual_Part returns the list of actual parameters passed to the instantiation P, or raises ASIS_Inappropriate_View if Is_Instance (P) returns False. Function Expanded_Body returns the expanded body of the instantiation P, or raises ASIS_Inappropriate_View if Is_Instance returns False.

23.3.2 compilation Units

```
function Is_Compilation_Unit (P : Program_Unit)
   return Boolean is abstract;

procedure Depends_Semantically_On (
   P : Program_Unit; Depends_On : out Program_Unit_Vector'Class) is abstract;

function Is_Subunit (P : Program_Unit) return Boolean is abstract;

function Stub_Of_Program_Unit (P : Program_Unit)
   return Asis.Declaration is abstract;

function Is_Library_Item (P : Program_Unit) return Boolean is abstract;
```

P specifies the program unit view declaration to query for each of these functions.

Function Is_Compilation_Unit returns True if and only if the program unit P is separately compiled as a compilation unit. Procedure Depends_Semantically_On returns in the Depends_On parameter the vector of other compilation units on which compilation unit P depends semantically, or raises ASIS_Inappropriate_View if Is_Compilation_Unit (P) returns False.

Function Is_Subunit returns True if and only if the program unit P is separately compiled as a subunit. Function Stub_Of_Program_Unit returns the Asis.Declaration that specifies the stub of the subunit P, or raises ASIS_Inappropriate_View if Is_Subunit (P) returns False. Function Is_Library_Item returns True if and only if the program unit P is separately compiled as a library item.

23.3.3 type Library_Item

```
type Library_Item is interface and Program_Unit;
```

The type Library_Item is used to represent program units that are separately compiled as library items — those for which Is_Library_Item returns True.

```
function Has_Parent_Library_Unit (L : Library_Item)
   return Boolean is abstract;

function Parent_Library_Unit (L : Library_Item)
   return Library_Item'Class is abstract;

function Is_Pure_Unit (L : Library_Item) return Boolean is abstract;

function Is_Preelaborated_Unit (L : Library_Item) return Boolean is abstract;

function Is_Remote_Call_Interface_Unit (L : Library_Item)
   return Boolean is abstract;
```

```
function Is_Remote_Types_Unit (L : Library_Item)
   return Boolean is abstract;
```

L specifies the library unit view declaration to query for each of these functions.

Function Has_Parent_Library_Unit returns True if and only if the library item L is a child of a unit other than package Standard. Function Parent_Library_Unit returns the parent library item of a child unit L, or raises ASIS_Inappropriate_View if Has_Parent_Library_Unit (L) returns False.

Function Is_Pure_Unit returns True if and only if the library item L is declared pure. Function Is_Preelaborated_Unit returns True if and only if the library item L is preelaborated. Function Is_Remote_Call_Interface_Unit returns True if and only if the pragma Remote_Call_Interface applies to the library item L. Function Is_Remote_Types_Unit returns True if and only if the pragma Remote_Types applies to the library item L.

23.3.4 Program Unit Vectors

```
package Program_Unit_Vectors is new
   Ada.Containers.Indefinite_Vectors (Positive, Program_Unit'Class);
type Program_Unit_Vector is new
   Program_Unit_Vectors.Vector with null record;
```

Type Program_Unit_Vector allows the declaration of a list of Program_Units.

23.4 package Asis.Subtype_Views

The library package Asis.Subtype_Views shall exist. The package shall provide interfaces equivalent to those described in the following subclauses.

23.4.1 type Subtype_View

```
type Subtype_View is interface and View;
```

The type Subtype_View represents a view of a subtype.

```
function Is_Elementary (S : Subtype_View) return Boolean is abstract;
function Is_Composite (S : Subtype_View) return Boolean is abstract;

function Is_Scalar (S : Subtype_View) return Boolean is abstract;
function Is_Enumeration (S : Subtype_View) return Boolean is abstract;
function Is_Boolean (S : Subtype_View) return Boolean is abstract;
function Is_Character (S : Subtype_View) return Boolean is abstract;
function Is_Numeric (S : Subtype_View) return Boolean is abstract;
function Is_Discrete (S : Subtype_View) return Boolean is abstract;
function Is_Integer (S : Subtype_View) return Boolean is abstract;
function Is_Signed_Integer (S : Subtype_View) return Boolean is abstract;
function Is_Modular_Integer (S : Subtype_View) return Boolean is abstract;
function Is_Real (S : Subtype_View) return Boolean is abstract;
function Is_Fixed_Point (S : Subtype_View) return Boolean is abstract;
function Is_Ordinary_Fixed_Point (S : Subtype_View)
   return Boolean is abstract;
function Is_Decimal_Fixed_Point (S : Subtype_View)
   return Boolean is abstract;
function Is_Floating_Point (S : Subtype_View) return Boolean is abstract;
function Is_Access (S : Subtype_View) return Boolean is abstract;
function Is_Access_To_Object (S : Subtype_View) return Boolean is abstract;
function Is_Access_To_Subprogram (S : Subtype_View)
   return Boolean is abstract;

function Is_Record (S : Subtype_View) return Boolean is abstract;
function Is_Record_Extension (S : Subtype_View) return Boolean is abstract;
function Is_Array (S : Subtype_View) return Boolean is abstract;
function Is_String (S : Subtype_View) return Boolean is abstract;
function Is_Protected (S : Subtype_View) return Boolean is abstract;
function Is_Task (S : Subtype_View) return Boolean is abstract;
function Is_Tagged (S : Subtype_View) return Boolean is abstract;
```

```
function Is_Formal_Subtype (S : Subtype_View) return Boolean is abstract;
function Is_Descended_From_Formal_Subtype (S : Subtype_View)
   return Boolean is abstract;

function Is_Constrained (S : Subtype_View) return Boolean is abstract;
function Is_Definite (S : Subtype_View) return Boolean is abstract;
```

The category and other characteristics of the subtype can be determined with the above functions. S specifies the view of a subtype to query for each of these functions.

23.4.2 Types, Subtypes, and Constraints

```
function Are_Of_Same_Type (Left, Right : Subtype_View)
   return Boolean is abstract;
```

Left and Right specify the views of subtypes to test.

Returns True if and only if Left and Right represent subtypes of the same type.

```
function Is_Unadorned_Subtype (S : Subtype_View) return Boolean is abstract;

function Unadorned_Subtype (S : Subtype_View)
   return Subtype_View'Class is abstract;

function Is_First_Subtype (S : Subtype_View) return Boolean is abstract;

function Is_Secondary_Subtype (S : Subtype_View) return Boolean is abstract;

function First_Subtype (S : Subtype_View)
   return Subtype_View'Class is abstract;

function Has_Constraint (S : Subtype_View) return Boolean is abstract;

function Constraint (S : Subtype_View)
   return Asis.Constraint is abstract;
```

S specifies the view of a subtype to query for each of these functions.

Function Is_Unadorned_Subtype returns True if the subtype S is scalar and the base subtype of its type, or the subtype S is tagged and the first subtype of its type, or the subtype S is neither tagged nor scalar, and the subtype has no constraint or null exclusion. Otherwise, Is_Unadorned_Subtype returns False.

Function Unadorned_Subtype returns a view of a subtype without any constraints or null exclusions. Specifically, if the subtype S is a scalar subtype, Unadorned_Subtype returns a view of the base subtype of S. If the subtype S is a tagged subtype, Unadorned_Subtype returns the first subtype of the type of S. If the subtype S is neither tagged nor scalar, Unadorned_Subtype returns a view of a subtype of the type of S that has no constraints or null exclusions.

Function Is_First_Subtype returns True if and only if the subtype S is the first subtype of its type. Function Is_Secondary_Subtype (S) is equivalent to not Is_First_Subtype (S). Function First_Subtype returns the a view of the first subtype for the type of the subtype S. If Is_First_Subtype (S) is True, First_Subtype returns S.

Function Has_Constraint returns True if and only if the subtype S has a constraint. Function Constraint returns the Asis element representing the constraint of the subtype S, or raises ASIS_Inappropriate_View if Has_Constraint (S) returns False.

NOTE The definition of Unadorned_Subtype corresponds to the meaning of the italicized T in the Ada Standard. The Ada Standard does not name this concept.

23.4.3 Static Subtypes and Constraints

```
function Is_Static_Subtype (S : Subtype_View) return Boolean is abstract;

function Is_Statically_Constrained (S : Subtype_View)
   return Boolean is abstract;
```

S specifies the view of a subtype to query for each of these functions.

Function Is_Static_Subtype returns True if and only if the subtype S is static, and returns False otherwise.

Function Is_Statically_Constrained returns True if and only if the subtype S is constrained and its constraint is static, and returns False otherwise.

```
function Are_Statically_Matching (S1, S2 : Subtype_View)
   return Boolean is abstract;
```

S1 and S2 specify views of subtypes to query.

Function Are_Statically_Matching returns True if and only if the subtypes S1 and S2 are of the same type and are statically matching, and returns False otherwise.

```
function Is_Statically_Compatible (S : Subtype_View;
   With_Subtype : Subtype_View) return Boolean is abstract;
```

S and With_Subtype specify views of subtypes to query.

Function Is_Statically_Compatible returns True if and only if the constraint of the subtype S is statically compatible with the subtype With_Subtype, and returns False otherwise.

23.4.4 Derived Types and Primitive Subprograms

```
procedure Primitive_Subprograms (S : Subtype_View;
   Primitives : out Declarative_Regions.View_Declaration_Vector'Class) is
abstract;

function Is_Derived (S : Subtype_View) return Boolean is abstract;

function Parent_Subtype (S : Subtype_View)
   return Subtype_View'Class is abstract;
```

S specifies the view of a subtype to query for each of these subprograms.

Function Primitive_Subprograms returns in the parameter Primitives a vector of view declarations for the primitive subprograms of the type of the subtype S. Function Is_Derived returns True if and only if the type of the subtype S is defined by a derived_type_definition, a private_extension_declaration, or a formal_derived_type_definition. Function Parent_Subtype returns the parent or ancestor subtype for the derived type S, or raises ASIS_Inappropriate_View if Is_Derived (S) returns False.

```
function Is_Descendant (S : Subtype_View; Of_Subtype : Subtype_View)
   return Boolean is abstract;
```

S and Of_Subtype specify the views of subtypes to compare.

Function Is_Descendant returns True if and only if the type of the subtype S is a descendant of the type of the subtype Of_Subtype.

```
function Ultimate_Ancestor (S : Subtype_View)
   return Subtype_View'Class is abstract;
```

S specifies the view of a subtype to query.

Function Ultimate_Ancestor returns an ancestor of the subtype S that is not itself a descendant of any other type; it will be the type itself if Is_Derived (S) returns False and Progenitors (S) (see 23.8.3) returns an empty vector. If there are multiple such ancestors, Ultimate_Ancestor returns the one that is not an interface type, if there is one. If all of the ultimate ancestors are interfaces, Ultimate_Ancestor returns an unspecified one.

NOTE Function Ultimate_Ancestor may stop on an incomplete or partial view. If the ultimate full view is needed, call Full_View on the result of Ultimate_Ancestor.

23.4.5 Incomplete and Partial Views

```
function Is_Incomplete_View (S : Subtype_View) return Boolean is abstract;
function Complete_View (S : Subtype_View)
   return Subtype_View'Class is abstract;
function Is_Partial_View (S : Subtype_View) return Boolean is abstract;
function Full_View (S : Subtype_View) return Subtype_View'Class is abstract;
```

S specifies the view of a subtype to query for all of these functions.

Function Is_Incomplete_View returns True if and only if the type of S is an incomplete view of a type.

If S is an incomplete view, Complete_View returns a view of the completion of S. Otherwise, Complete_View returns S.

Function Is_Partial_View returns True if and only if the type of S is a partial view of a type.

Function Full_View returns the full view of S. If S is already a full view, Full_View returns S.

NOTE The Full_View of an incomplete type gives the full view of the completion. This is even true for an incomplete view declared by a limited view of a private type. However, the Complete_View of such an incomplete view is the private type.

23.4.6 Subtype Aspects

```
function Subtype_Size (S : Subtype_View) return ASIS_Natural is abstract;
function Subtype_Alignment (S : Subtype_View) return ASIS_Natural is abstract;
```

S specifies the view of a subtype to query for both of these functions.

Function Subtype_Size returns the same value as the Size attribute of the subtype S, if the subtype is elementary or if Is_Aspect_Specified (S, Size) returns True. The result is implementation-defined in other cases, and may be ASIS_Natural'Last. Function Subtype_Alignment returns the same value as the Alignment attribute of the subtype S.

23.4.7 Subtype Holders

```
package Subtype_Holders is new Ada.Containers.Indefinite_Holders
(Subtype_View'Class);
type Subtype_Holder is new Subtype_Holders.Holder with null record;
```

Type Subtype_Holder allows the declaration of an uninitialized variable (including a component) that can hold a Subtype_View.

23.4.8 Subtype Vectors

```
package Subtype_Vectors is new
   Ada.Containers.Indefinite_Vectors (Positive, Subtype_View'Class);
type Subtype_Vector is new Subtype_Vectors.Vector with null record;
```

Type Subtype_Vector allows the declaration of a list of Subtype_Views.

23.5 package Asis.Object_Views

The library package Asis.Object_Views shall exist. The package shall provide interfaces equivalent to those described in the following subclauses.

23.5.1 type Object_View

```
type Object_View is interface and View;
```

The type Object_View represents views of objects and values, as denoted by names or expressions.

```
function Is_Constant_View (O : Object_View) return Boolean is abstract;

function Nominal_Subtype (O : Object_View)
   return Subtype_View'Class is abstract;
```

O specifies the view of an object to query for both of these functions.

Function Is_Constant_View returns True if and only if the object O is a constant view of an object, or is a view of a value. Function Nominal_Subtype returns the nominal subtype of the view.

23.5.2 Components of Objects

```
function Is_Component (O : Object_View) return Boolean is abstract;

function Enclosing_Object (O : Object_View)
   return Object_View'Class is abstract;

function Is_Indexed_Component (O : Object_View) return Boolean is abstract;

function Index_Value (O : Object_View; Dimension : Positive := 1)
   return Object_View'Class is abstract;

function Is_Selected_Component (O : Object_View) return Boolean is abstract;

function Selector_Declaration (O : Object_View)
   return View_Declaration'Class is abstract;

function Is_Component_Selected_Component (O : Object_View) return Boolean is
abstract;

function Position (O : Object_View) return Object_View'Class is abstract;

function First_Bit (O : Object_View) return Object_View'Class is abstract;

function Last_Bit (O : Object_View) return Object_View'Class is abstract;
```

O specifies the view of an object to query for each of these functions.

Function Is_Component returns True if and only if the view O is of a component of an enclosing composite object. Function Enclosing_Object returns a view of the enclosing object of the component O, or raises ASIS_Inappropriate_View if Is_Component (O) returns False.

Function Is_Indexed_Component returns True if and only if the view O is of an indexed component. Function Index_Value returns a view of the value of the index of the indexed component O, or raises ASIS_Inappropriate_View if Is_Indexed_Component (O) returns False.

Function Is_Selected_Component returns True if and only if the view O is of a selected component. Function Selector_Declaration returns the declaration of the selected component O, or raises ASIS_Inappropriate_View if Is_Selected_Component (O) returns False.

Function Is_Component_Selected_Component returns True if and only if the view O is of a selected component that denotes a component. Functions Position, First_Bit, and Last_Bit, return a view of the value of a reference to the corresponding attribute of the given component O, or raise ASIS_Inappropriate_View if Is_Component_Selected_Component (O) returns False.

23.5.3 Aliased Views and Dereferences

```
function Is_Aliased (O : Object_View) return Boolean is abstract;

type Static_Accessibility_Level is range 0 .. <implementation-defined>;

Library_Level : constant Static_Accessibility_Level := 0;
Deepest_Accessibility_Level : constant Static_Accessibility_Level;
Incomparable_Accessibility_Level : constant Static_Accessibility_Level;

function Static_Accessibility (O : Object_View)
   return Static_Accessibility_Level is abstract;

function Is_Dereference (O : Object_View) return Boolean is abstract;

function Dereferenced_Value (O : Object_View)
   return Object_View'Class is abstract;

function Is_Implicit_Dereference (O : Object_View) return Boolean is abstract;
```

O specifies the view of an object to query for each of these functions.

Function Is_Aliased returns True if and only if the view O is aliased. Function Static_Accessibility returns the static accessibility level of the view O, or raises ASIS_Inappropriate_View if Is_Aliased (O) returns False. The static accessibility level value 0 is returned if the view O is of a library level object. Incomparable_Accessibility_Level is returned if the view O is a dereference of an access-to-object parameter. Deepest_Accessibility_Level is returned if the view O is a dereference of an access-to-subprogram parameter.

Function Is_Dereference returns True if and only if the object O is a dereference of an access-to-object value. Function Dereferenced_Value returns a view of the access value that was dereferenced in the dereference O, or raises ASIS_Inappropriate_View if Is_Dereference (O) returns False. Function Is_Implicit_Dereference returns True if and only if the object O is an implicit dereference of an access-to-object value.

23.5.4 Static Values

```
type Longest_Discrete is range <implementation-defined>;

function Is_Static_Discrete (O : Object_View) return Boolean is abstract;

function Static_Discrete_Value (O : Object_View)
   return Longest_Discrete is abstract;

function Static_Discrete_Image (O : Object_View)
   return Wide_String is abstract;

type Longest_Float is digits <implementation-defined>;

function Is_Static_Real (O : Object_View) return Boolean is abstract;

function Static_Real_Value (O : Object_View)
   return Longest_Float is abstract;

function Static_Real_Image (O : Object_View)
   return Wide_String is abstract;

function Is_Static_String (O : Object_View) return Boolean is abstract;

function Static_String_Value (O : Object_View)
   return Wide_String is abstract;
```

O specifies the view of an object to query for each of these functions.

Function Is_Static_Discrete returns True if and only if the view O is a view of the value of a static expression of a discrete type. Function Static_Discrete_Value returns the position number of the value of the static discrete expression O, or raises ASIS_Inappropriate_View if Is_Static_Discrete (O) returns False. If the position number of the value is outside the range of Longest_Discrete, it raises Constraint_Error. Function Static_Discrete_Image returns the image of the position number of the value of the static discrete expression O, with the syntax used by the Image attribute of its type (with a

leading minus if negative, and a leading space otherwise), or raises ASIS_Inappropriate_View if Is_Static_Discrete (O) returns False. A correct image is returned even if the position number of the value is outside the base range of the type.

Function Is_Static_Real returns True if and only if the view O is of the value of a static expression of a real type. Function Static_Real_Value returns the value of the static real expression O converted to the Longest_Float type, or raises ASIS_Inappropriate_View if Is_Static_Real (O) returns False. If the value is outside the range of Longest_Real, it raises Constraint_Error. Function Static_Real_Image returns a string containing an optional minus sign, followed by a pair of non-negative integers separated by the character '/' corresponding to a numerator and a denominator for the reduced rational representation of the exact absolute value of the static real expression O. The denominator is one when the numerator is zero. Static_Real_Image raises ASIS_Inappropriate_View if Is_Static_Real (O) returns False.

Function Is_Static_String returns True if and only if the view O is of the value of a static expression of a string type. Function Static_String_Value returns a Wide_String whose characters match the corresponding characters of the static value of the string expression O (encoded in UTF-16, as described in Section 3). The function raises ASIS_Inappropriate_View if Is_Static_String (O) returns False.

NOTE 1 Static_Discrete_Value can be used on integer, character, and enumeration values.

NOTE 2 The result of Static_Real_Value may not have all of the precision of the original static value even if it is in range. If the exact value is important, use Static_Real_Image to retrieve the value instead of Static_Real_Value.

23.5.5 Representational Object Attributes

```
function Object_Size (O : Object_View) return ASIS_Natural is abstract;
function Object_Alignment (O : Object_View) return ASIS_Natural is abstract;
```

O specifies the view of an object to query for both of these functions.

Function Object_Size returns the same value as the Size attribute of the object O, if the object is elementary, if Is_Aspect_Specified (O, Size) returns True, or if its subtype's size is specified. The result is implementation-defined in other cases, and may be ASIS_Natural'Last. Function Object_Alignment returns the same value as the Alignment attribute of the object O. If the view is of a value rather than an object, the result of Object_Size and Object_Alignment is implementation-defined, but is within the range of values possible for objects of the same type.

23.5.6 Object Holders

```
package Object_Holders is new Ada.Containers.Indefinite_Holders
(Object_View'Class);
type Object_Holder is new Object_Holders.Holder with null record;
```

Type Object_Holder allows the declaration of an uninitialized variable (including a component) that can hold a Object_View.

23.6 package Asis.Profiles

The library package Asis.Profiles shall exist. The package shall provide interfaces equivalent to those described in the following subclauses.

23.6.1 type Profile

```
type Profile is private;
```

The type Profile specifies the profile of a callable entity.

23.6.2 function Parameters (Profile)

```
function Parameters (P : Profile) return Declarative_Regions.Region_Part;
```

P specifies the Profile to query.

Returns the Region_Part containing the parameters of the profile.

If there are no parameters in the profile, returns a Region_Part for which Is_Empty returns true.

23.6.3 Function queries

```
function Is_Function (P : Profile) return Boolean;
function Result_Subtype (P : Profile) return Subtype_View'Class;
```

P specifies the Profile to query for both of these functions.

Is_Function returns True if the profile represents a function, and returns False otherwise.

Result_Subtype returns the Subtype_View of the return part of the profile, if Is_Function (P) is True. Otherwise, Result_Subtype raises ASIS_Inappropriate_View.

23.6.4 Family index queries

```
function Has_Family_Index (P : Profile) return Boolean;
function Family_Index_Subtype (P : Profile) return Subtype_View'Class;
```

P specifies the Profile to query for both of these functions.

Has_Family_Index returns True if the Profile applies to an entry family, and returns False otherwise.

Family_Index_Subtype returns the entry index subtype of the profile P if Has_Family_Index (P) is True. Otherwise, Family_Index_Subtype raises ASIS_Inappropriate_View.

23.6.5 Profile conventions

```
function Convention (P : Profile) return Conventions;
function Convention_Identifier (P : Profile) return Wide_String;
```

P specifies the Profile to query for both of these functions.

Function Convention returns the Convention of Profile P. Function Convention_Identifier returns "Intrinsic", "Ada", "Protected", or "Entry" when function Convention returns the corresponding value of type Conventions. When Convention returns Other_Convention, Convention_Identifier returns the identifier given in the pragma Convention, Import, or Export used to specify the convention of the entity.

23.7 package Asis.Subtype_Views.Elementary

The library package Asis.Subtype_Views.Elementary shall exist. The package shall provide interfaces equivalent to those described in the following subclauses.

23.7.1 Elementary subtypes

```
type Elementary_Subtype is interface and Subtype_View;
```

The type Elementary_Subtype represents a view of an elementary subtype.

```
function Is_Universal (E : Elementary_Subtype) return Boolean is abstract;
```

E specifies the Elementary_Subtype to query.

Returns True if and only if the type of the given subtype is universal_integer, universal_real, universal_fixed, or universal_access. Returns False otherwise.

23.7.2 Scalar subtypes

```
type Scalar_Subtype is interface and Elementary_Subtype;
```

The type Scalar_Subtype represents a view of a scalar subtype.

```
function Base_Subtype (S : Scalar_Subtype)
   return Scalar_Subtype'Class is abstract;

function Low_Bound (S : Scalar_Subtype) return Object_View'Class is abstract;
function High_Bound (S : Scalar_Subtype) return Object_View'Class is abstract;

function Is_Root_Numeric (S : Scalar_Subtype) return Boolean is abstract;
```

S specifies the view of a scalar subtype to query for each of these functions.

Function Base_Subtype returns a view of the base subtype of the type of the subtype S. Functions Low_Bound and High_Bound return a view of the corresponding bound of the scalar subtype S. If the scalar subtype S is unconstrained, Low_Bound and High_Bound return a view of the corresponding bound of the base range of the type of the subtype.

Function Is_Root_Numeric returns True if the type of S is root_integer or root_real, and returns False otherwise.

23.7.3 Discrete subtypes

```
type Discrete_Subtype is interface and Scalar_Subtype;
```

The type Discrete_Subtype represents a view of a discrete subtype.

23.7.4 Access subtypes

```
type Access_Subtype is interface and Elementary_Subtype;
```

The type Access_Subtype represents a view of an access subtype.

```
function Is_Anonymous_Access (A : Access_Subtype) return Boolean is abstract;

function Is_Access_Parameter (A : Access_Subtype) return Boolean is abstract;

function Is_Access_Result (A : Access_Subtype) return Boolean is abstract;

function Is_Access_Discriminant (A : Access_Subtype)
   return Boolean is abstract;

function Static_Accessibility (A : Access_Subtype)
   return Object_Views.Static_Accessibility_Level is abstract;

function Excludes_Null (A : Access_Subtype) return Boolean is abstract;
```

A specifies the view of an access subtype to query for each of these functions.

Function Is_Anonymous_Access returns True if and only if the type of the subtype A is defined by an access_definition rather than an access_type_definition. Function Is_Access_Parameter returns True if and only if the type of the subtype A is that of an access parameter. Function Is_Access_Result returns True if and only if the type of the subtype A is the result type of a function with an access result. Function Is_Access_Discriminant returns True if and only if the type of the subtype A is that of an access discriminant.

Function Static_Accessibility returns the static accessibility level of the type of the subtype A. The static accessibility level value 0 is returned if the type is a library level type. Incomparable_Accessibility_Level is returned for the type of an access-to-object parameter. Deepest_Accessibility_Level is returned for the type of an access-to-

subprogram parameter. Function Excludes_Null returns True if and only if the subtype A excludes null.

23.7.5 Access-to-object subtypes

```
type Access_To_Object_Subtype is interface and Access_Subtype;
```

The type Access_To_Object_Subtype represents a view of an access-to-object subtype.

```
function Designated_Subtype (A : Access_To_Object_Subtype)
   return Subtype_View'Class is abstract;
function Is_Access_To_Constant (A : Access_To_Object_Subtype)
   return Boolean is abstract;
function Is_Pool_Specific (A : Access_To_Object_Subtype)
   return Boolean is abstract;
function Storage_Pool (A : Access_To_Object_Subtype)
   return Object_View'Class is abstract;
function Storage_Size (A : Access_To_Object_Subtype)
   return Object_View'Class is abstract;
```

A specifies the Access_To_Object_Subtype to query for each of these functions.

Function Designated_Subtype returns a view of the designated subtype of the access-to-object subtype A. Function Is_Access_To_Constant returns True if and only if the type of the subtype A is an access-to-constant type. Function Is_Pool_Specific returns True if and only if the type of the subtype A is pool specific.

Function Storage_Pool returns a view of the object denoted by the Storage_Pool attribute of the type of the subtype A, if Is_Aspect_Specified (A, Storage_Pool) returns True. The result is implementation-defined in other cases.

Function Storage_Size returns a view of a value equal to the Storage_Size of the type of the subtype A, if the type is defined to have zero Storage_Size, or if Is_Aspect_Specified (A, Storage_Size) returns True. The result is implementation-defined in other cases.

23.7.6 Access-to-subprogram subtypes

```
type Access_To_Subprogram_Subtype is interface and Access_Subtype;
```

The type Access_To_Subprogram_Subtype represents a view of an access-to-subprogram subtype.

```
function Designated_Profile (A : Access_To_Subprogram_Subtype)
   return Profile is abstract;
```

A specifies the view of an access-to-subprogram subtype to query.

Returns a view of the designated profile of the type of the subtype A.

23.8 package Asis.Subtype_Views.Composite

The library package Asis.Subtype_Views.Composite shall exist. The package shall provide interfaces equivalent to those described in the following subclauses.

23.8.1 Composite Subtypes

```
type Composite_Subtype is interface and Subtype_View;
```

The type Composite_Subtype represents a view of a composite subtype.

```
function Is_Limited (C : Composite_Subtype) return Boolean is abstract;
function Contains_Task (C : Composite_Subtype) return Boolean is abstract;
```

```
function Needs_Finalization (C : Composite_Subtype)
  return Boolean is abstract;

function Has_Preelaborable_Initialization (C : Composite_Subtype)
  return Boolean is abstract;

function Has_Unknown_Discriminants (C : Composite_Subtype)
  return Boolean is abstract;

function Has_Known_Discriminants (C : Composite_Subtype)
  return Boolean is abstract;

function Discriminants (C : Composite_Subtype)
  return Declarative_Regions.Region_Part is abstract;

function Discriminants_Have_Defaults (C : Composite_Subtype)
  return Boolean is abstract;

function Has_Nondiscriminant_Region_Parts (C : Composite_Subtype)
  return Boolean is abstract;

function Nondiscriminant_Region_Parts (C : Composite_Subtype)
  return Declarative_Regions.Region_Part_List is abstract;
```

C specifies the view of a composite subtype to query for each of these functions.

Function Is_Limited returns True if and only if the view of the subtype C is limited. Function Contains_Task returns True if and only if the type of the subtype C has a part that is a task type. Function Needs_Finalization returns True if and only if the type of the subtype C needs finalization. Function Has_Preelaborable_Initialization returns True if and only if the subtype C has preelaborable initialization.

Function Has_Unknown_Discriminants returns True if and only if the subtype C has unknown discriminants. Function Has_Known_Discriminants returns True if and only if the subtype C has known discriminants. Function Discriminants returns a Region_Part for the discriminant part of the subtype C, or raises ASIS_Inappropriate_View if the subtype view does not have known discriminants. Function Discriminants_Have_Defaults returns True if and only if the subtype C has known discriminants with default_expressions.

Function Has_Nondiscriminant_Region_Parts returns True if and only if the subtype C is a descendant of a record type, a record extension, a task type, or a protected type. Function Nondiscriminant_Region_Parts returns a list of Region_Parts, one for each separate visible region part, each comprising components, entries, and protected subprograms from a single list of components or items of subtype C. The returned list is empty if Has_Nondiscriminant_Region_Parts (C) is False.

23.8.2 Array Subtypes

```
type Array_Subtype is interface and Composite_Subtype;
```

The type Array_Subtype represents a view of an array subtype.

```
function Component_Subtype (A : Array_Subtype)
   return Subtype_View'Class is abstract;

function Num_Dimensions (A : Array_Subtype) return Positive is abstract;

function Index_Subtype (A : Array_Subtype; Dimension : Positive := 1)
   return Elementary.Discrete_Subtype'Class is abstract;

function Is_String_Subtype (A : Array_Subtype) return Boolean is abstract;
```

A specifies the view of an array subtype to query for each of these functions.

Function Component_Subtype returns a view of the component subtype of the type of the array subtype A. Function Num_Dimensions returns a count of the number of dimensions of the type of the array subtype A. Function Index_Subtype returns a view of the Dimension-th index subtype of the type of the array subtype A. Function Is_String_Subtype returns True if and only if the type of the subtype A is a string type.

23.8.3 Tagged Subtypes

```
type Tagged_Subtype is interface and Composite_Subtype;
```

The type Tagged_Subtype represents a view of a tagged subtype.

```
function Is_Interface (T : Tagged_Subtype) return Boolean is abstract;
function Is_Abstract (T : Tagged_Subtype) return Boolean is abstract;

function Is_Synchronized_Tagged (T : Tagged_Subtype)
   return Boolean is abstract;

function Is_Classwide (T : Tagged_Subtype) return Boolean is abstract;

function Root_Subtype (T : Tagged_Subtype)
  return Tagged_Subtype'Class is abstract;

function Is_Specific (T : Tagged_Subtype) return Boolean is abstract;

function Classwide_Subtype (T : Tagged_Subtype)
  return Tagged_Subtype'Class is abstract;

procedure Progenitors (
  T : Tagged_Subtype;
  Progenitors : out Tagged_Subtype_Vector'Class) is abstract;

function External_Tag (T : Tagged_Subtype) return String is abstract;
```

T specifies the view of a tagged subtype to query for each of these subprograms.

Function Is_Interface returns True if and only if the type of the subtype T is an interface. Function Is_Abstract returns True if and only if the type of the subtype T is abstract. Function Is_Synchronized_Tagged returns True if and only if the type of the subtype T is a synchronized tagged type. Function Is_Classwide returns True if and only if the type of the subtype T is classwide.

Function Root_Subtype returns a view of a specific subtype *S* that is the root of the class T (that is, *S*'Class = T), or raises ASIS_Inappropriate_View if the type of the subtype T is not classwide. Function Is_Specific returns True if and only if the type of the subtype T is a specific tagged type. Function Classwide_Subtype returns a view of the classwide subtype rooted at the subtype T, equivalent to the Class attribute of the subtype T, or raises ASIS_Inappropriate_View if the subtype T is not a specific tagged subtype. Function Progenitors returns a vector of the progenitors, if any, of the type of the subtype T. The result is an empty vector if the type has no progenitors. Function External_Tag returns a view of the string value equal to the External_Tag of the type of the subtype T, if External_Tag is specified. Otherwise, the result is implementation-defined.

23.8.4 Tagged Subtype Vectors

```
package Tagged_Subtype_Vectors is new
   Ada.Containers.Indefinite_Vectors (Positive, Tagged_Subtype'Class);
type Tagged_Subtype_Vector is new
   Tagged_Subtype_Vectors.Vector with null record;
```

Type Tagged_Subtype_Vector allows the declaration of a list of Tagged_Subtypes.

23.9 package Asis.Callable_Views

The library package Asis.Callable_Views shall exist. The package shall provide interfaces equivalent to those described in the following subclauses.

23.9.1 type Callable_View

```
type Callable_View is interface and View;
```

The type Callable_View represents views of callable entities, as denoted by names or expressions.

23.9.2 function Callable_Profile

```
function Callable_Profile (C : Callable_View) return Profile is abstract;
```

C specifies the callable view to query.

Returns the Profile of the callable view C.

23.9.3 Callable view categorization

```
function Is_Subprogram (C : Callable_View) return Boolean is abstract;
function Is_Enumeration_Literal (C : Callable_View)
   return Boolean is abstract;
function Is_Procedure (C : Callable_View) return Boolean is abstract;
function Is_Entry (C : Callable_View) return Boolean is abstract;
function Is_Function (C : Callable_View) return Boolean is abstract;
function Is_Abstract (C : Callable_View) return Boolean is abstract;
function Is_Null (C : Callable_View) return Boolean is abstract;
```

C specifies the callable view to query for each of these functions.

Is_Subprogram returns True if the callable view C is of a subprogram (that is, C has Callable_View_kinds of A_Noninstance_Subprogram, A_Subprogram_Instance, A_Subprogram_Renaming, A_Protected_Subprogram, An_Imported_Subprogram, An_Attribute_Subprogram, An_Intrinsic_Subprogram, A_Designated_Subprogram, or A_Generic_Formal_Subprogram), and returns False otherwise.

Is_Enumeration_Literal returns True if the callable view C is of an enumeration literal, and returns False otherwise.

Is_Procedure returns True if the callable view C denotes a procedure, and returns False otherwise.

Is_Entry returns True if the callable view C denotes an entry (that is, C has Callable_View_kinds of A_Protected_Entry or A_Task_Entry), otherwise it returns False.

Is_Function returns True if the callable view C denotes a function, and returns False otherwise.

Is_Abstract returns True if the callable view C denotes a subprogram declared by an abstract_subprogram_declaration or an abstract inherited subprogram, and returns False otherwise.

Is_Null returns true if the Callable_View denotes a Null Procedure, and returns False otherwise.

NOTE A use of an enumeration literal is formally a function call, so a view of such a use is a Callable_View for which Is_Function returns True.

23.9.4 Primitive operations

```
function Is_Primitive (C : Callable_View) return Boolean is abstract;
```

C specifies the callable view to query.

Returns True if the callable view C is of a primitive subprogram. Returns False otherwise.

```
procedure Primitive_On_Subtypes (
   C : Callable_View; Subtypes : out Subtype_Vector'Class) is abstract;
```

C specifies the callable view to query.

Subtype_Vector is a vector of the subtypes returned by the query.

Returns a vector of the subtypes on which the subprogram C is primitive. For a callable view C that is not primitive, returns Subtype_Vector(Subtype_Vectors.Empty_Vector).

```
function Is_Dispatching_Operation (C : Callable_View)
   return Boolean is abstract;
```

C specifies the callable view to query.

Returns True if C denotes a dispatching operation. Returns False otherwise.

```
function Associated_Tagged_Type (C : Callable_View)
   return Composite.Tagged_Subtype'Class is abstract;
```

C specifies the callable view to query.

Returns the controlling tagged type of a dispatching operation C. If C is not a dispatching operation, ASIS_Inappropriate_View is raised.

23.9.5 Prefixed views

```
function Is_Prefixed_View (C : Callable_View) return Boolean is abstract;
```

C specifies the callable view to query.

Returns True if C is a prefixed view of a subprogram. Returns False otherwise.

```
function Prefix_Object (C : Callable_View)
   return Object_View'Class is abstract;
```

C specifies the callable view to query.

If Is_Prefixed_View (C) is True, returns the Object_View of the of the prefix of C. Otherwise, raises ASIS_Inappropriate_View.

```
function Unprefixed_Callable_View (C : Callable_View)
   return Callable_View'Class is abstract;
```

C specifies the callable view to query.

If Is_Prefixed_View (C), returns the standard (unprefixed) form callable_view of C. Returns C otherwise.

23.9.6 Access-to-subprogram views

```
function Is_Designated_Subprogram (C : Callable_View)
   return Boolean is abstract;
```

C specifies the callable view to query.

Returns True if the callable view C represents an access to subprogram call; otherwise, returns False.

```
function Access_To_Subprogram_Value (C : Callable_View)
   return Object_View is abstract;
```

C specifies the callable view to query.

If Is_Designated_Subprogram (C) is True, returns the Object_View of the access_to_subprogram object associated with the callable view C.

Otherwise, raises ASIS_Inappropriate_View.

```
function Is_Implicit_Dereference (C : Callable_View)
   return Boolean is abstract;
```

C specifies the callable view to query.

Is_Implicit_Dereference returns True if and only if the callable view C is an implicit dereference of an access-to-subprogram value.

23.9.7 Callable view holders

```
package Callable_Holders is new Ada.Containers.Indefinite_Holders
(Callable_View'Class);
type Callable_Holder is new Callable_Holders.Holder with null record;
```

Type Callable_Holder allows the declaration of an uninitialized variable (including a component) that can hold a Callable_View.

23.10 package Asis.Object_Views.Access_Views

The library package Asis.Object_Views.Access_Views shall exist. The package shall provide interfaces equivalent to those described in the following subclauses.

23.10.1 type Access_Object_View

```
type Access_Object_View is interface and Object_View;
```

The type Access_Object_View represents views of access objects and values, as denoted by names or expressions.

23.10.2 Designated object operations

```
function Is_Object_Access_Attribute_Reference (O : Access_Object_View) return
Boolean is abstract;

function Designated_Object (O : Access_Object_View)
   return Object_View'Class is abstract;

function Is_Implicit_Access_Attribute_Reference (O : Access_Object_View)
   return Boolean is abstract;

function Is_Subprogram_Access_Attribute_Reference (O : Access_Object_View)
   return Boolean is abstract;

function Designated_Subprogram (O : Access_Object_View)
   return Callable_View'Class is abstract;
```

O specifies the view of an access object to query for each of these functions.

Function Is_Object_Access_Attribute_Reference returns True if and only if the view O is of an (explicit or implicit) Access or Unchecked_Access attribute reference of an aliased object. Function Designated_Object returns the object designated by the access value in the attribute O, or raises ASIS_Inappropriate_View if Is_Object_Access_Attribute_Reference (O) returns False.

Function Is_Implicit_Access_Attribute_Reference returns True if and only if the view O is of a value produced by an implicit Access attribute reference as part of a call on a prefixed view of a subprogram, where the first parameter of the unprefixed subprogram is an access parameter.

Function Is_Subprogram_Access_Attribute_Reference returns True if and only if the view O is of an (explicit or implicit) Access attribute reference of a subprogram. Function Designated_Subprogram returns a view of the subprogram designated by the access value in the attribute O, or raises ASIS_Inappropriate_View if Is_Subprogram_Access_Attribute_Reference (O) returns False.

23.11 package Asis.Package_Views

The library package Asis.Package_Views shall exist. The package shall provide interfaces equivalent to those described in the following subclauses.

23.11.1 type Package View

```
type Package_View is interface and View;
```

The type Package_View represents a view of a package. Every package has a non-limited view. A package may also have a limited view.

23.11.2 function Has_Limited_View

```
function Has_Limited_View (P : Package_View) return Boolean is abstract;
```

P specifies the package view to query.

Returns True if a limited view of P exists (including if P is a limited view). Returns False otherwise.

23.11.3 function Is_Limited_View

```
function Is_Limited_View (P : Package_View) return Boolean is abstract;
```

P specifies the package view to query.

Returns True if P denotes the limited view of a package. Returns False otherwise.

23.11.4 function Full_View

```
function Full_View (P : Package_View) return Package_View'Class
  is abstract;
```

P specifies the package view to query.

If Is_Limited_View (P) then returns the corresponding full view of P. In this case, Limited_View (Full_View (P)) = P.

Returns P otherwise.

NOTE In all cases, Is_Limited_View (Full_View (P)) = False.

23.11.5 function Is_Full_View

```
function Is_Full_View (P : Package_View) return Boolean is abstract;
```

P specifies the package view to query.

Returns True if P denotes the full view of a package. Returns False otherwise.

NOTE In all cases, Is_Full_View (P) /= Is_Limited_View (P).

23.11.6 function Limited_View

```
function Limited_View (P : Package_View)
   return Package_View'Class is abstract;
```

P specifies the package view to query.

If Is_Limited_View (P) is True, then returns P. Otherwise, if Has_Limited_View (P) is True, then returns the corresponding limited view of P. If Has_Limited_View (P) is False, then raises ASIS_Inappropriate_View.

23.11.7 function Is_Formal_Package

```
function Is_Formal_Package (P : Package_View) return Boolean is abstract;
```

P specifies the package view to query.

Returns True if P denotes a formal package. Returns False otherwise.

23.11.8 Part selectors

```
function Visible_Part (P : Package_View)
   return Declarative_Regions.Region_Part is abstract;
function Private_Part (P : Package_View)
   return Declarative_Regions.Region_Part is abstract;
```

P specifies the package view to query.

Visible_Part yields the appropriate (either limited or full) view of the visible part of the package. Private_Part yields the appropriate (either limited or full) view of the private part of the package.

NOTE To find the body of a package view P, use Body_Part (Defined_Region (P)). This technique can also be used to access the stubs of a package.

23.11.9 Package Holders

```
package Package_Holders is new Ada.Containers.Indefinite_Holders
(Package_View'Class);
type Package_Holder is new Package_Holders.Holder with null record;
```

Type Package_Holder allows the declaration of an uninitialized variable (including a component) that can hold a Package_View.

23.12 package Asis.Generic_Views

The library package Asis.Generic_Views shall exist. The package shall provide interfaces equivalent to those described in the following subclauses.

23.12.1 type Generic_View

```
type Generic_View is interface and View;
```

The type Generic_View represents a view of a generic unit.

23.12.2 function Generic_Formal_Part (view)

```
function Generic_Formal_Part (G : Generic_View)
   return Declarative_Regions.Region_Part is abstract;
```

G specifies the view of a generic to query.

Yields a Region_Part that specifies the formal part of the generic G.

23.12.3 function Is_Generic_Package

```
function Is_Generic_Package (G : Generic_View) return Boolean
   is abstract;
```

G specifies the view of a generic to query.

Returns True if G denotes a generic package. Returns False otherwise.

23.12.4 function Current_Package_Instance

```
function Current_Package_Instance (G : Generic_View)
   return Package_Views.Package_View'Class is abstract;
```

G specifies the view of a generic to query.

If Is_Generic_Package (G) is False, then raises ASIS_Inappropriate_View.

As seen from within itself, a generic unit is an instance (that is, the current instance of the given generic); it is not a generic unit. This function yields a view of that instance for generic G.

23.12.5 function Is_Generic_Subprogram

```
function Is_Generic_Subprogram (G : Generic_View) return Boolean
   is abstract;
```

G specifies the view of a generic to query.

Returns True if G denotes a generic subprogram. Returns False otherwise.

NOTE In all cases, Is_Generic_Subprogram (G) /= Is_Generic_Package (G).

23.12.6 function Current_Subprogram_Instance

```
function Current_Subprogram_Instance (G : Generic_View)
   return Callable_Views.Callable_View'Class is abstract;
```

G specifies the view of a generic to query.

If Is_Generic_Subprogram (G) is False, then raises ASIS_Inappropriate_View.

As seen from within itself, a generic unit is an instance (that is, the current instance of the given generic); it is not a generic unit. This function yields a view of that instance for generic G.

23.12.7 Generic Holders

```
package Generic_Holders is new Ada.Containers.Indefinite_Holders
(Generic_View'Class);
type Generic_Holder is new Generic_Holders.Holder with null record;
```

Type Generic_Holder allows the declaration of an uninitialized variable (including a component) that can hold a Generic_View.

23.13 package Asis.Exception_Views

The library package Asis.Exception_Views shall exist. The package shall provide interfaces equivalent to those described in the following subclauses.

23.13.1 type Exception_View

```
type Exception_View is interface and View;
```

The type Exception_View represents a view of an exception.

NOTE To determine whether a Discard_Names pragma applies to an exception view E, use Is_Aspect_Specified (E, Discarded_Names).

23.13.2 Exception Holders

```
package Exception_Holders is new Ada.Containers.Indefinite_Holders
(Exception_View'Class);
type Exception_Holder is new Exception_Holders.Holder with null record;
```

Type Exception_Holder allows the declaration of an uninitialized variable (including a component) that can hold a Exception_View.

23.14 package Asis.Statement_Views

The library package Asis.Statement_Views shall exist. The package shall provide interfaces equivalent to those described in the following subclauses.

23.14.1 type Statement_View

```
type Statement_View is interface and View;
```

The type Statement_View represents a view of a statement.

23.14.2 function Corresponding_Statement

```
function Corresponding_Statement (S : Statement_View)
   return ASIS.Statement is abstract;
```

S specifies the statement view to query.

Yields the ASIS.Statement value corresponding to the statement S.

23.14.3 Statement Holders

```
package Statement_Holders is new Ada.Containers.Indefinite_Holders
(Statement_View'Class);
type Statement_Holder is new Statement_Holders.Holder with null record;
```

Type Statement_Holder allows the declaration of an uninitialized variable (including a component) that can hold a Statement_View.

23.15 package Asis.Declarations.Views

The library package Asis.Declarations.Views shall exist. The package shall provide interfaces equivalent to those described in the following subclauses.

23.15.1 function Corresponding_View_Declaration

```
function Corresponding_View_Declaration (Declaration: in Asis.Declaration)
   return Asis.Views.Declarative_Regions.View_Declaration'Class;
```

Declaration specifies the declaration to query.

Returns a view that specifies the entity denoted by Declaration.

Declaration expects an element that has the following Element_Kinds:

A_Declaration

Raises ASIS_Inappropriate_Element with a Status of Value_Error for any element that does not have one of these expected kinds.

23.16 package Asis.Definitions.Views

The library package Asis.Definitions.Views shall exist. The package shall provide interfaces equivalent to those described in the following subclauses.

23.16.1 function Corresponding_Subtype_View

```
function Corresponding_Subtype_View (Type_Definition : in Asis.Type_Definition)
   return Asis.Subtype_Views.Subtype_View'Class;
```

Type_Definition specifies the type definition to query.

Returns a view that specifies the subtype denoted by Type_Definition.

Type_Definition expects an element that has one of the following Definition_Kinds:

A_Type_Definition
A_Private_Type_Definition
A_Tagged_Private_Type_Definition
A_Private_Extension_Definition
A_Task_Definition
A_Protected_Definition
A_Formal_Type_Definition

Raises ASIS_Inappropriate_Element with a Status of Value_Error for any element that does not have one of these expected kinds.

23.17 package Asis.Expressions.Views

The library package Asis.Expressions.Views shall exist. The package shall provide interfaces equivalent to those described in the following subclauses.

23.17.1 function Corresponding_View

```
function Corresponding_View (Expression : in Asis.Expression)
   return Asis.Views.View'Class;
```

Expression specifies the expression to query.

Returns the view that describes the semantic meaning of the Expression.

Expression expects an element that has the following Element_Kinds:

An_Expression

Raises ASIS_Inappropriate_Element with a Status of Value_Error for any element that does not have one of these expected kinds.

NOTE The returned view denotes the expression, and not the declaration of the expression. In addition, the view represents the information available to the expression. This latter point means that the information in the view may depend on the visibility at the point of the reference that the view denotes. For instance, if the Expression denotes a private type whose full type is not visible at the point of Expression in the program and Corresponding_View (Expression) returns V, Is_Record (V) will be False even if the full type of the private type is a record type.

Annex A
(informative)
Glossary

All terms which are defined in the Ada Standard as Ada technical terms are used in the ASIS specification in full accordance with those definitions. See the Ada International Standard for definitions.

Ada Standard. Collectively, the International Standard ISO/IEC 8652:1995(E) as corrected by ISO/IEC 8652:1995/COR1:2001(E) and amended by ISO/IEC 8652:1995/AMD.1:2007(E).

Ancestor. Ancestors of a library unit are itself, its parent, its parent's parent, and so on. (Standard is an ancestor of every library unit).

ASIS. is used in reference to the acronym Ada Semantic Interface Specification. It is also used in reference to the Ada package Asis.

ASIS implementation. All the hardware and software that implement the ASIS specification for a given Ada implementation and that provide the functionality required by the ASIS specification.

ASIS queries. Those subprogram interfaces (and only those) defined in the ASIS standard; these are supported by types, subtypes, and exceptions also defined in the ASIS standard. Thus, ASIS queries and supporting entities are together the ASIS interface. The following informal query classification is used by the ASIS community: "black-box" queries are those ASIS queries which produce information about compilation units and "white-box" queries are those ASIS queries which produce information about Elements; semantic queries are those ASIS queries which express semantic properties of ASIS Elements in terms of other Elements; structural queries are those ASIS queries which provide the top-down decomposition and reverse bottom-up composition of the compilation unit according to its syntax structure. (Note that semantic queries are generally named "Corresponding_..." or "Implicit_..." in the ASIS specification.)

CASE. Computer Assisted Software/System Engineering refers to the methods dedicated to an engineering discipline for the development of information systems together with automated tools that can be used in this process.

Closure. A term commonly used instead of needed units.

Compilation unit. “The term *compilation unit* is used to refer to a compilation_unit. When the meaning is clear from context, the term is also used to refer to the library_item of a compilation_unit or to the proper_body of a subunit”;. [The Ada Standard, 10.1.1(9)]. ASIS says "ASIS compilation unit" when the intent is to stress that the ASIS viewpoint on an Ada compilation unit is described in the ASIS standard. Note that the term "compilation unit" can refer to either syntactical category "compilation_unit" or to the library_item of a compilation_unit or to the proper_body of a subunit (that is, the compilation_unit without the context_clause and the separate (parent_unit_name)).

Compilation_Unit [type]. An ASIS private type whose values denote an Ada compilation unit or configuration pragma from the environment denoted by some open ASIS context. A non-nil value of the Compilation_Unit type also contains information about some physical object from the "external world" treated by the underlying Ada implementation as the corresponding Ada compilation unit or as a result of compiling a configuration pragma.

Container. Logical collection of ASIS compilation units. For example, some container can hold compilation units which include Ada predefined types, another container can hold implementation-defined packages. Containers provide the implementation-defined way of grouping the compilation units accessible for an ASIS application through the ASIS queries.

Container [type]. An ASIS private type whose values denote a set of compilation units being a subset of the set of compilation units making up a context.

Context. A set of compilation units and configuration pragmas processed by an ASIS application. ASIS provides any information from a context by treating this set as if its elements make up an environment declarative part by modeling some view (most likely one of the views of the underlying Ada implementation) on the environment. ASIS may process several different contexts at a time.

Context [type]. An ASIS private type whose values denote a set of compilation units considered by ASIS as making up an Ada environment declarative part from which to provide information.

Dependent. Dependents of a compilation unit are all the compilation units that depend semantically on it, either directly or indirectly. A is a dependent of B, if B is a supporter of A.

Descendants. Descendants of a library unit relation are the inverse of the ancestor relation.

DII. Dynamic Invocation Interface is an API which allows dynamic construction of CORBA object invocations. It is used at compile time when a client does not have knowledge about the object it wants to invoke. With this interface an argument list is marshalled, a function is named, and a request for service is sent to the object server. DII implementations will usually have an asynchronous mode of operation.

Element. A common abstraction used by ASIS to denote the syntax components (both explicit and implicit) of ASIS compilation units. The term Element is also used as the synonym for "the value of the ASIS Element type". See also "Explicit element" and "Implicit element".

Element [type]. An ASIS private type, whose values represent the syntax components (both explicit and implicit) of ASIS compilation units.

Environment. “Each compilation unit submitted to the compiler is compiled in the context of an environment declarative_part (or simply environment), which is a conceptual declarative_part that forms the outermost declarative region of the context of any compilation. At run time, an environment forms the declarative_part of the body of the environment task of a partition.” [ISO/IEC 8652:1995(E), 10.1.4(1)]. Note that the mechanisms for creating an environment and for adding and replacing compilation units within an environment are implementation-defined.

Explicit element. An ASIS Element, representing a language construct that appears explicitly in the program text for the compilation unit (e.g., an explicit declaration).

Extension. Non-standard facilities (other library units, non-standard children of standard ASIS library units, subprograms, etc.) which provide additional information from ASIS types, or modify the behavior of otherwise standard ASIS facilities to provide alternative or additional functionality.

Family. The family of a given unit is defined as the set of compilation units that comprise the given unit's declaration, body, descendants, and subunits (and subunits of subunits and descendants, etc.).

Id. A way of identifying a particular element, from a particular compilation unit, from a particular context.

Id [type]. An ASIS private type implementing the Id abstraction. The values of this type can be written to files. These values can be read back from files and converted into values of the Element type with the use of a suitable open context.

IDL. Interface Definition Language is used by CORBA to specify the interfaces that objects will present to the outside world. CORBA then specifies a mapping from IDL to a specific implementation language like C++ or Java. Standard mappings exist for Ada, C, C++, Lisp, Smalltalk, Java, COBOL, PL/I and Python. There are also non-standard mappings for Perl, Visual Basic, Ruby, Erlang, and Tcl implemented by object request brokers (ORBs) written for those languages.

IEC. International Electrotechnical Commission is a not-for-profit, non-governmental international standards organization that prepares and publishes International Standards for all electrical, electronic and related technologies collectively known as "electrotechnology".

Implementor. A company, institution, or other group (such as a vendor) who develops an ASIS implementation; thus an ASIS implementor. There are also Ada implementors, who provide Ada compilation systems; and there are ASIS-based tool (or, ASIS Application) implementors, who develop tools which are based upon the ASIS standard.

Implicit element. An ASIS Element, representing a language construct that does not exist in the program text for the compilation unit, but could occur at a given place in the program text as a consequence of the semantics of another construct, (e.g., an implicit declaration, a generic instantiation).

ISO. International Organization for Standardization (Organisation internationale de normalisation), widely known as ISO, is an international standard-setting body composed of representatives from various national standards organizations.

JTC. ISO/IEC JTC is the Joint Technical Committee of the International Organization for Standardization (ISO) and the International Electrotechnical Commission (IEC). It deals with all matters of information technology.

Line. The logical representation of a line of text from the source code of the external representation of a compilation unit.

Line [type]. An ASIS private type for the ASIS Line abstraction. The values of the Line type represent the lines of text from the source code of the external representation of compilation units.

Needed Units. The set of compilation units ultimately needed by the given compilation unit to make up or to be included in a completed partition.

Optional functionality. The subset of ASIS facilities that are explicitly identified in the ASIS standard as optional which may legitimately be omitted from a Basic Conforming ASIS implementation, but shall be included in any Fully Conforming ASIS implementation, unless stated otherwise in the ASIS specification.

ORB. A CORBA Object Request Broker is a piece of middleware software that allows programmers to make program calls from one computer to another via a network.

Queries. See ASIS queries.

Relation (between ASIS Compilation Units). Semantic relationships between compilation units (as discussed in chapter 10 of the Ada Standard). The Relation_Kinds type enumerates the kinds of relations that can exist between compilation units. See also Dependent, Extended Family, and Supporter.

Required functionality. The subset of ASIS facilities which are not explicitly identified in the ASIS standard as optional which shall be included in a Basic or Fully Conforming ASIS implementation, unless stated otherwise in the ASIS specification.

Semantic queries. See ASIS queries.

Structural queries. See ASIS queries.

Supporter. Supporters of a compilation unit are units on which it semantically depends, either directly or indirectly. B is a supporter of A, if A is a dependent of B.

Annex B
(informative)
ASIS application examples

B.1 An ASIS application to traverse a compilation unit

The following example of an ASIS tool prompts the user for the name of an Ada package specification, traverses that compilation unit, and prints all explicit declarations with their kind.

```
with Asis;                                                          -- 3
with Asis.Errors;                                                   -- 4
with Asis.Exceptions;                                               -- 5
with Asis.Implementation;                                           -- 6
with Asis.Ada_Environments;                                         -- 8
with Asis.Compilation_Units;                                        -- 10
with Asis.Elements;                                                 -- 13
with Asis.Iterator;                                                 -- 14
with Asis.Declarations;                                             -- 15
with Ada.Wide_Text_Io; use Ada.Wide_Text_Io;

procedure ASIS_Application_Example is

   My_Context                 : Asis.Context;                       -- 3.3
   My_Unit                    : Asis.Compilation_Unit;              -- 3.8
   Unit_Name                  : Wide_String ( 1 .. 100 );
   Unit_Name_Length           : Natural;

   procedure Report_Declarations (Unit : in Asis.Compilation_Unit) is -- 3.8

      My_Element              : Asis.Element;                       -- 3.4
      My_Control              : Asis.Traverse_Control               -- 3.11
                                   := Asis.Continue;
      My_State                : Boolean := True;

      procedure Process_Element
                   (Elem      : in     Asis.Element;                -- 3.4
                    Control   : in out Asis.Traverse_Control;       -- 3.11
                    State     : in out Boolean);

      procedure No_Op
                   (Elem      : in     Asis.Element;                -- 3.4
                    Control   : in out Asis.Traverse_Control;       -- 3.11
                    State     : in out Boolean);

      procedure Find_and_Print_Declarations is new
                    Asis.Iterator.Traverse_Element                  -- 14.1
                       (Boolean, Process_Element, No_Op);

      procedure No_Op
                   (Elem      : in     Asis.Element;                -- 3.4
                    Control   : in out Asis.Traverse_Control;       -- 3.11
                    State     : in out Boolean) is
      begin
         null;
      end No_Op;

      procedure Process_Element
                   (Elem      : in     Asis.Element;                -- 3.4
                    Control   : in out Asis.Traverse_Control;       -- 3.11
                    State     : in out Boolean) is

         package Kind_Io is new Ada.Wide_Text_Io.Enumeration_Io
                             (Asis.Declaration_Kinds);              -- 3.7.4

         Decl_Kind : Asis.Declaration_Kinds :=                      -- 3.7.4
             Asis.Elements.Declaration_Kind (Elem);                 -- 13.9

      begin -- Process_Element

         case Decl_Kind is

            when Asis.Not_A_Declaration => null;                    -- 3.7.4

            when others =>
```

```
                if not Asis."="                                          -- 3.7.6
                          (Asis.Elements.Declaration_Origin (Elem),      -- 13.19
                           Asis.An_Explicit_Declaration) then            -- 3.7.6
                   return;
                end if;

                declare

                   Name_List : Asis.Defining_Name_List                   -- 3.6
                          := Asis.Declarations.Names (Elem);             -- 15.1

                begin
                   for I in Name_List'Range loop
                       Put (Asis.Declarations.Defining_Name_Image
                           (Name_List (I)));                             -- 15.2
                       Put (" (is kind) ");
                       Kind_Io.Put (Decl_Kind);
                       New_Line;
                   end loop;
                end;

          end case;

       end Process_Element;

    begin -- Report_Declarations
       My_Element   := Asis.Elements.Unit_Declaration (Unit);            -- 13.1
       Find_and_Print_Declarations (My_Element, My_Control, My_State);
    end Report_Declarations;

begin -- ASIS_Application_Example

    Asis.Implementation.Initialize;                                      -- 6.6
    Asis.Ada_Environments.Associate(My_Context, "My Context");           -- 8.3
    Asis.Ada_Environments.Open (My_Context);                             -- 8.4

    Put_Line ("Type the name of an Ada package specification");
    Get_Line (Unit_Name, Unit_Name_Length);

    My_Unit := Asis.Compilation_Units.Library_Unit_Declaration           -- 10.6
                  ( Unit_Name ( 1 .. Unit_Name_Length), My_Context );

    if Asis.Compilation_Units.Is_Nil (My_Unit)                           -- 10.15
    then
       Put ("Context does not contain the requested unit: ");
       Put (Unit_Name ( 1 .. Unit_Name_Length));
       New_Line;
    else
       Put ("Context contains the requested unit: ");
       Put (Unit_Name ( 1 .. Unit_Name_Length));
       New_Line;
       Report_Declarations ( My_Unit );
       New_Line;
    end if;

    Asis.Ada_Environments.Close (My_Context);                            -- 8.5
    Asis.Ada_Environments.Dissociate (My_Context);                       -- 8.6
    Asis.Implementation.Finalize;                                        -- 6.8

exception

    when   Asis.Exceptions.ASIS_Inappropriate_Context                    -- 5
         | Asis.Exceptions.ASIS_Inappropriate_Container                  -- 5
         | Asis.Exceptions.ASIS_Inappropriate_Compilation_Unit           -- 5
         | Asis.Exceptions.ASIS_Inappropriate_Element                    -- 5
         | Asis.Exceptions.ASIS_Inappropriate_Line                       -- 5
         | Asis.Exceptions.ASIS_Inappropriate_Line_Number                -- 5
         | Asis.Exceptions.ASIS_Failed                                   -- 5
        =>

         Put (Asis.Implementation.Diagnosis);                            -- 6.10
         New_Line;
         Put ("Status Value is ");
         Put (Asis.Errors.Error_Kinds'Wide_Image                         -- 4.1
                        (Asis.Implementation.Status));                   -- 6.9
         New_Line;

    when others =>

         Put_Line ("ASIS Application failed because of non-ASIS reasons");

end ASIS_Application_Example;
```

Sample input for the ASIS Application Example is the following package specification named asis_test:

```
package asis_test is
   type T is ( A, B, C);
   S : integer := T'BASE'SIZE;
end asis_test;
```

Result of executing ASIS Application Example:

```
Type the name of an Ada package specification
asis_test
Context contains the requested unit: asis_test
asis_test (is kind) A_PACKAGE_DECLARATION
T (is kind) AN_ORDINARY_TYPE_DECLARATION
A (is kind) AN_ENUMERATION_LITERAL_SPECIFICATION
B (is kind) AN_ENUMERATION_LITERAL_SPECIFICATION
C (is kind) AN_ENUMERATION_LITERAL_SPECIFICATION
S (is kind) A_VARIABLE_DECLARATION
```

B.2 An ASIS application to build a call tree

This example prints call tree information (i.e., a list of all procedure, function, and entry calls made within a compilation unit) for each compilation unit in the context. The output format is of the form:

<Calling_Compilation_Unit> (calls) <Called_Program_Unit> at line <Line_Number> where:

<Calling_Compilation_Unit> is the Expanded Name of the Unit making the call <Called_Program_Unit> is the name of the program unit being called <Line_Number> is the first line number of the call in the source file

```
with Asis;                                                          -- 3
with Asis.Errors;                                                   -- 4
with Asis.Exceptions;                                               -- 5
with Asis.Implementation;                                           -- 6
with Asis.Ada_Environments;                                         -- 8
with Asis.Compilation_Units;                                        -- 10
with Asis.Elements;                                                 -- 13
with Asis.Iterator;                                                 -- 14
with Asis.Declarations;                                             -- 15
with Asis.Expressions;                                              -- 17
with Asis.Statements;                                               -- 18
with Asis.Text;                                                     -- 20

with Ada.Wide_Text_Io; use Ada.Wide_Text_Io;

procedure ASIS_Call_Tree_Example is

   My_Context              : Asis.Context;                          -- 3.3

   procedure No_Op
               (Elem        : in     Asis.Element;                 -- 3.4
                Control     : in out Asis.Traverse_Control;        -- 3.11
                State       : in out Boolean);

   procedure Report_Calls
              (An_Element: in     Asis.Element;                    -- 3.4
               Control   : in out Asis.Traverse_Control;           -- 3.11
               State     : in out Boolean);

   procedure Print_Call_Tree is new
               Asis.Iterator.Traverse_Element                      -- 14.1
                  (Boolean, Report_Calls, No_Op);

   procedure No_Op
               (Elem        : in     Asis.Element;                 -- 3.4
                Control     : in out Asis.Traverse_Control;        -- 3.11
                State       : in out Boolean) is
   begin
      null;
   end No_Op;
```

```
procedure Output_Call (Caller : Asis.Element;                          -- 3.4
                       Callee : Asis.Declaration) is                   -- 3.6

   Calling_Cu   : Asis.Compilation_Unit;                               -- 3.8
   Calling_Unit : Asis.Declaration;                                    -- 3.6

begin -- Output_Call

   Calling_Cu := Asis.Elements.Enclosing_Compilation_Unit (Caller); -- 13.2

   if Asis.Compilation_Units.Is_Nil (Calling_Cu) then                  -- 10.15
      Put ("An_Unknown_Unit");
   else
      Put (Asis.Compilation_Units.Unit_Full_Name (Calling_Cu));        -- 10.19
   end if;

   Put (" (calls) ");
   Put (Asis.Declarations.Defining_Name_Image                          -- 15.2
                  (Asis.Declarations.Names (Callee) (1)));             -- 15.1
   Put (" at line ");
   Put (Asis.Text.Line_Number'Wide_Image                              -- 20.2
                (Asis.Text.First_Line_Number (Caller)));              -- 20.6
   New_Line;

end Output_Call;

procedure Report_Calls (An_Element : in Asis.Element;                  -- 3.4
                        Control    : in out Asis.Traverse_Control;     -- 3.11
                        State      : in out Boolean) is

   Callee : Asis.Declaration;                                          -- 3.6

begin -- Report_Calls

   case Asis.Elements.Element_Kind (An_Element) is                     -- 13.6

      when Asis.An_Expression =>                                       -- 3.7.1
         case Asis.Elements.Expression_Kind (An_Element) is            -- 13.32
            when Asis.A_Function_Call =>                               -- 3.7.19

               Callee := Asis.Expressions.Corresponding_Called_Function
                                 (An_Element);                         -- 17.29

               if not Asis.Elements.Is_Nil (Callee) then               -- 13.40
                  Output_Call (An_Element, Callee);
               end if;

            when others =>
               null;
         end case;

      when Asis.A_Statement =>                                         -- 3.7.1
         case Asis.Elements.Statement_Kind (An_Element) is             -- 13.36

            when Asis.A_Procedure_Call_Statement |                     -- 3.7.22
                 Asis.An_Entry_Call_Statement =>                       -- 3.7.22

               Callee := Asis.Statements.Corresponding_Called_Entity
                             (An_Element);                             -- 18.28

               if not Asis.Elements.Is_Nil (Callee) then               -- 13.40
                  Output_Call (An_Element, Callee);
               end if;

            when others =>
               null;
         end case;

      when others =>
         null;

   end case;

end Report_Calls;

procedure Process_Units (Unit_List : in Asis.Compilation_Unit_List) is -- 3.9

   Control : Asis.Traverse_Control := Asis.Continue;                   -- 3.11
   State   : Boolean := True;

begin

   for I in Unit_List'Range loop
```

```
         case Asis.Compilation_Units.Unit_Origin (Unit_List (I)) is     -- 10.3
            when Asis.An_Application_Unit =>                            -- 3.10.3
               New_Line;
               Put_Line ("Processing Unit: " &
                                 Asis.Compilation_Units.Unit_Full_Name
                                    (Unit_List (I)));                   -- 10.19
               Print_Call_Tree (Asis.Elements.Unit_Declaration          -- 13.1
                                    (Unit_List (I)), Control, State);
            when others =>
               null;
         end case;

      end loop;

   end Process_Units;

begin -- ASIS_Call_Tree_Example

   Asis.Implementation.Initialize;                                      -- 6.6
   Asis.Ada_Environments.Associate(My_Context, "My Context");           -- 8.3
   Asis.Ada_Environments.Open (My_Context);                             -- 8.4

   Process_Units (Asis.Compilation_Units.Compilation_Units (My_Context)); --
10.10

   Asis.Ada_Environments.Close (My_Context);                            -- 8.5
   Asis.Ada_Environments.Dissociate (My_Context);                       -- 8.6
   Asis.Implementation.Finalize;                                        -- 6.8

exception

   when Asis.Exceptions.ASIS_Inappropriate_Context                      -- 5
        | Asis.Exceptions.ASIS_Inappropriate_Container                  -- 5
        | Asis.Exceptions.ASIS_Inappropriate_Compilation_Unit           -- 5
        | Asis.Exceptions.ASIS_Inappropriate_Element                    -- 5
        | Asis.Exceptions.ASIS_Inappropriate_Line                       -- 5
        | Asis.Exceptions.ASIS_Inappropriate_Line_Number                -- 5
        | Asis.Exceptions.ASIS_Failed                                   -- 5
      =>

      Put (Asis.Implementation.Diagnosis);                              -- 6.10
      New_Line;
      Put ("Status Value is ");
      Put (Asis.Errors.Error_Kinds'Wide_Image                           -- 4.1
                     (Asis.Implementation.Status));                     -- 6.9
      New_Line;

   when others =>

      Put_Line ("Asis Application failed because of non-ASIS reasons");

end ASIS_Call_Tree_Example;
```

Consider the context containing the following compilation units:

```
package P is

   procedure P1;
   procedure P2;
   procedure P3(X : integer);
   function F1 return integer;

end;

package body P is

   procedure P1 is separate;
   procedure P2 is separate;
   procedure P3(X : integer) is separate;
   function F1 return integer is separate;

begin
   P1;
end;

separate (P)
function F1 return integer is
begin
   return 0;
end;
```

```
separate (P)
procedure P1 is
   x : integer := F1;
begin
   P2;
   P3(x);
end;

separate (P)
procedure P2 is
begin
   P3(F1);
end;

separate (P)
procedure P3(X : integer) is
begin
   null;
end;
```

Applying ASIS_Call_Tree_Example to the context given above yields the following output:

```
Processing Unit: P

Processing Unit: P
P (calls) P1 at line  9

Processing Unit: P.F1

Processing Unit: P.P1
P.P1 (calls) F1 at line  3
P.P1 (calls) P2 at line  5
P.P1 (calls) P3 at line  6

Processing Unit: P.P2
P.P2 (calls) P3 at line  4
P.P2 (calls) F1 at line  4

Processing Unit: P.P3
```

Annex C
(informative)
Miscellaneous ASIS I/O and IDL approaches

This Annex contains examples of miscellaneous approaches to deal with I/O and IDL issues. The first two portions contain packages for application I/O of ASIS types: Portable_Data and Id. The third portion contains an approach to providing an ASIS IDL. Annex C consists of:

- C.1, “package Asis.Ids.Id_Io” — I/O for type Id from Asis.Ids
- C.2, “Using ASIS with CORBA and IDL”

C.1 package Asis.Ids.Id_Io

```
with Asis;
with Asis.Ids;
with Ada.Direct_Io;
with Ada.Io_Exceptions;
package Id_Io is
------------------------------------------------------------------------------
Asis.Ids.Id_Io provides Id I/O Facilities.
------------------------------------------------------------------------------
This interface is a copy of the Ada.Direct_Io interface.
The internals of this package are implementation dependent and the
amount of space taken, in an Id file, by an ASIS Id value is variable.
An ASIS Id value has a fixed 'Size, but, the size of the data represented
by that Id value can be arbitrarily large. It can contain access values.
------------------------------------------------------------------------------
Annex C.1.1  type File_Type
------------------------------------------------------------------------------

    type File_Type is limited private;

------------------------------------------------------------------------------
Annex C.1.2  type File_Mode
------------------------------------------------------------------------------

    type File_Mode is (In_File,
                       Inout_File,
                       Out_File);

    -- Count has an ASIS implementation-defined upper bound.
    -- Note: An Id can take up more than one "slot" in an Id file.
    -- The 1st Id in a file will probably be at From => 1, but, the 2nd
    -- Id in a file is not necessarily at From => 2.

------------------------------------------------------------------------------
Annex C.1.3  type Count
------------------------------------------------------------------------------

    type Count is new Integer range 0 ..
        Asis.Implementation_Defined_Integer_Constant;

------------------------------------------------------------------------------
Annex C.1.4  type Positive_Count
------------------------------------------------------------------------------

    subtype Positive_Count is Count range 1 .. Count'Last;

------------------------------------------------------------------------------
Annex C.1.5  procedure Create
------------------------------------------------------------------------------

    procedure Create (File      : in out File_Type;
                      Mode      : in     File_Mode := Inout_File;
                      Name      : in     Wide_String    := "";
                      Form      : in     Wide_String    := "");

------------------------------------------------------------------------------
Annex C.1.6  procedure Open
------------------------------------------------------------------------------
```

```
    procedure Open (File        : in out File_Type;
                    Mode        : in     File_Mode;
                    Name        : in     Wide_String;
                    Form        : in     Wide_String := "");
-------------------------------------------------------------------------------
Annex C.1.7  procedure Close
-------------------------------------------------------------------------------
    procedure Close (File       : in out File_Type);
-------------------------------------------------------------------------------
Annex C.1.8  procedure Delete
-------------------------------------------------------------------------------
    procedure Delete (File      : in out File_Type);
-------------------------------------------------------------------------------
Annex C.1.9  procedure Reset
-------------------------------------------------------------------------------
    procedure Reset (File       : in out File_Type; Mode : File_Mode);
    procedure Reset (File       : in out File_Type);
-------------------------------------------------------------------------------
Annex C.1.10 function Mode
-------------------------------------------------------------------------------
    function Mode (File         : in File_Type) return File_Mode;
-------------------------------------------------------------------------------
Annex C.1.11 function Name
-------------------------------------------------------------------------------
    function Name (File         : in File_Type) return Wide_String;
-------------------------------------------------------------------------------
Annex C.1.12 function Form
-------------------------------------------------------------------------------
    function Form (File         : in File_Type) return Wide_String;
-------------------------------------------------------------------------------
Annex C.1.13 function Is_Open
-------------------------------------------------------------------------------
    function Is_Open (File      : in File_Type) return Boolean;
-------------------------------------------------------------------------------
Annex C.1.14 procedure Read
-------------------------------------------------------------------------------
Read operations
    procedure Read (File        : in  File_Type;
                    Item        : out Asis.Ids.Id;
                    From        : in  Positive_Count);
    procedure Read (File        : in  File_Type;
                    Item        : out Asis.Ids.Id);
    procedure Read (File        : in  File_Type;
                    Item        : out Asis.Element;
                    From        : in  Positive_Count;
                    Context     : in  Asis.Context);
    procedure Read (File        : in  File_Type;
                    Item        : out Asis.Element;
                    Context     : in  Asis.Context);
-------------------------------------------------------------------------------
Annex C.1.15 procedure Write
-------------------------------------------------------------------------------
Write operations
   procedure Write (File        : in File_Type;
                    Item        : in Asis.Ids.Id;
                    To          : in Positive_Count);
    procedure Write (File       : in File_Type;
                     Item       : in Asis.Ids.Id);
    procedure Write (File       : in File_Type;
                     Item       : in Asis.Element;
                     To         : in Positive_Count);
    procedure Write (File       : in File_Type;
                     Item       : in Asis.Element);
```

```
------------------------------------------------------------------------------
Annex C.1.16 procedure Set_Index
------------------------------------------------------------------------------

    procedure Set_Index (File   : in File_Type;
                         To     : in Positive_Count);

------------------------------------------------------------------------------
Annex C.1.17 function Index
------------------------------------------------------------------------------

    function Index (File        : in File_Type) return Positive_Count;

------------------------------------------------------------------------------
Annex C.1.18 function Size
------------------------------------------------------------------------------

    function Size (File         : in File_Type) return Count;

------------------------------------------------------------------------------
Annex C.1.19 function End_Of_File
------------------------------------------------------------------------------

    function End_Of_File (File  : File_Type) return Boolean;

------------------------------------------------------------------------------
Annex C.1.20 Exceptions
------------------------------------------------------------------------------

    -- Exceptions

    Status_Error : exception renames Ada.Io_Exceptions.Status_Error;
    Mode_Error   : exception renames Ada.Io_Exceptions.Mode_Error;
    Name_Error   : exception renames Ada.Io_Exceptions.Name_Error;
    Use_Error    : exception renames Ada.Io_Exceptions.Use_Error;
    Device_Error : exception renames Ada.Io_Exceptions.Device_Error;
    End_Error    : exception renames Ada.Io_Exceptions.End_Error;
    Data_Error   : exception renames Ada.Io_Exceptions.Data_Error;

private
    -- ASIS implementation dependent.
    package Ids_Id_Io is new Ada.Direct_Io (Asis.Ids.Id);
    type File_Type is new Ids_Id_Io.File_Type;

------------------------------------------------------------------------------

end Id_Io;
```

C.2 Using ASIS with CORBA and IDL

An ASIS application can use the required ASIS interfaces within a Common Object Request Broker Architecture (CORBA) networked environment using the Interface Definition Language (IDL).

Figure C-1 depicts how the ASIS CORBA clients and servers can be created from the ASIS API. ASIS in IDL can be obtained by reverse engineering the ASIS API expressed in this International Standard. This can be done either manually or through automated tools. Once the ASIS in IDL is created, CORBA clients and servers can be created via the automatic generation of IDL described in the remaining portion of this annex.

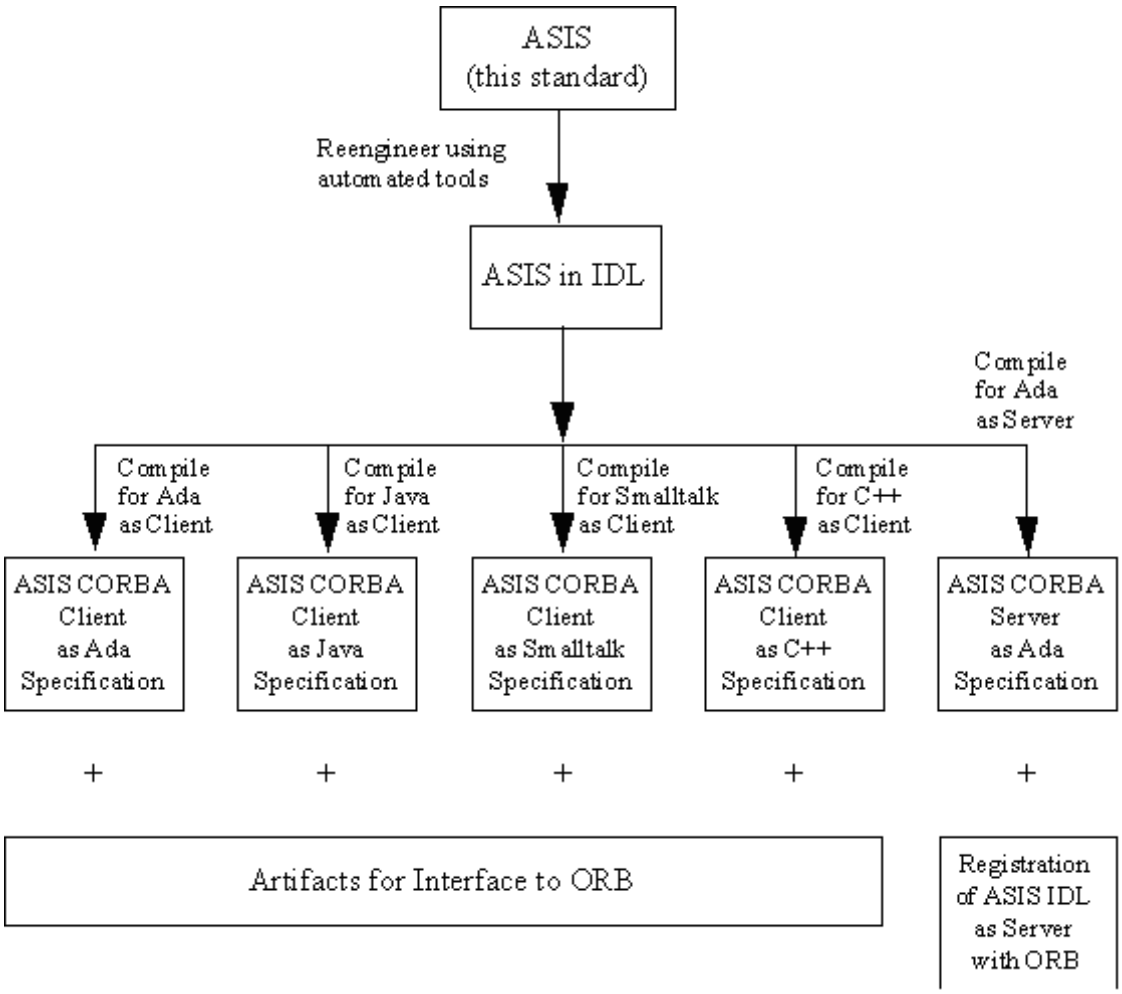

Figure C.1 – Generation of client/server ASIS artifacts

C.2.1 ASIS API server

When compiled on the server side using the Ada option, an Ada specification is created with the necessary objects to register the ASIS IDL with the ORB. This specification should be nearly identical to the original ASIS specification. The ASIS specification created by the compilation of the ASIS IDL should map to the original ASIS specification supported by the implementor. In this way, an implementor's ASIS interface can be used locally on the host node or through the CORBA client / server mechanism as desired by the tool implementor. ASIS implementors should be able to easily take advantage of an IDL client / server capability with little additional work.

C.2.2 ASIS API client tool

The ASIS IDL can be compiled on the client side producing an Ada package specification, C++ code, or Smalltalk code. This allows tools in any language to access the ASIS interface. Assuming the Ada compilation option, a logical Ada package body executable is created having hooks to the ASIS server via the ORB. If the Ada compilation option is used, then tools developed using Ada could access this Ada specification as if the ASIS package body were locally available. Any client tool can now use the ASIS CORBA client as an Ada specification through an Ada with clause. Client tools in Java, in C++, or in Smalltalk can also be easily built.

C.2.3 Implementing generic subprogramTraverse_Element in a CORBA environment

An IDL must first be created from the ASIS specification (this International Standard). Automatic IDL generators can create IDL from Ada. Language features such as generics are problematic for this automatic conversion as generics cannot be automatically translated into IDL. The package Asis.Iterator is the only required ASIS package containing a generic. The following steps should be performed to create an IDL to support the required ASIS interfaces:

a) Remove child package Asis.Iterator.

b) Remove the optional packages (Asis.Data_Decomposition and Asis.Data_Decomposition.Portable_Transfer).

c) Perform a global replace of "Asis." with "ASIS_CORBA."

d) Rename package Asis as ASIS_CORBA.

e) Submit package ASIS_CORBA to an automatic IDL generation process.

f) Insert the attached IDL segments for Interface Iterator (see C.2.4) and Interface Generics (see C.2.4) to restore the effects of Asis.Iterator into the automatically generated IDL. [Note: the Interface Element and enum Traverse_Control could be automatically generated; they are shown here as they are supporters of Interface Generics and Interface Iterator.]

g) After the IDL is compiled with Ada as the target language, the ASIS_CORBA package will appear as shown in the attached Ada code.

h) On the Client side, a client can use package Asis directly when the specification has been mapped into ASIS_CORBA. The type Asis.Element will map into the package ASIS_CORBA.Element (there will also be the package Asis.Elements mapping into the package ASIS_CORBA.Elements). The package Asis.Iterator will map into ASIS_CORBA.Iterator. The generics are placed into a separate IDL module called Interface Generics which will create the package Asis.Generics. This will allow clients to map My_Pre_Operation and My_Post_Operation procedures into the Pre_Operation and Post_Operation procedures of Asis.Generics. This mapping can be achieved by:

- Implementing the ASIS_CORBA.Generics in a separate server and pass the reference to the Generics object from the ASIS client to the ASIS server.
- Using a multithreaded CORBA implementation with an additional server thread in the client.
- Implementing the client mapping to accept callbacks in the client (not portable).

C.2.4 IDL to implement generic subprogram Traverse_Element

It is possible to automatically generate IDL from the ASIS specification. The following interfaces for Interface Generics and Interface Iterator should be manually inserted into the automatically generated IDL prior to compilation. Compilation of the resulting IDL will produce the necessary artifacts to register the IDL with the Object Request Broker (ORB) and the output Ada depicted in C.2.5.

```
module ASIS_CORBA
{
    // ...
    interface Element { };         // This is the private type Asis.Element
     enum Traverse_Control {       // This type is found in package Asis
```

```
            Continue,               // Continues the normal depth-first traversal.
            Abandon_Children,       // Prevents traversal of the current element's
                                    //   children.
            Abandon_Siblings,       // Prevents traversal of the current element's
                                    // children and remaining siblings.
            Terminate_Immediately}; // Does exactly that.
        interface Generics          // abstract
        {                           // Generic portion of
    Asis.Iterator.Traverse_Control

                void Pre_Operation
                    (in    Element           This_Element,
                     inout Traverse_Control  What_Next);

                void Post_Operation
                    (in    Element            This_Element,
                     inout Traverse_Control   What_Next);

                                    // no state info needed, children will add
    state
           };
        interface Iterator          // Abstract
        {                           // This is Asis.Iterator.Traverse_Element

                void Traverse_Element
                     (in    Element            Starting_From,
                      inout Traverse_Control   What_Next,
                      in    Traversal          Using);
          };
          // ...
    };
```

C.2.5 Ada code output by the IDL compiler

The following code segment represents the Ada specification generated by an IDL compiler for the IDL code presented in Annex C.3.4 when targeted for an Ada output.

```
with CORBA.Object;
package ASIS_CORBA is
    -- …
    type Traverse_Control is       -- This type is found in package Asis
        (Continue,                 -- Continues the normal depth-first traversal.
          Abandon_Children,        -- Prevents traversal of the current element's
                                   -- children.
          Abandon_Siblings,        -- Prevents traversal of the current element's
                                   -- children and remaining siblings.
          Terminate_Immediately); -- Does exactly that.
    -- …
    package Element is             -- This is the private type Asis.Element

        type Ref is new CORBA.Object.Ref with null record;
        -- Narrow/Widen functions
        function To_Ref (From : CORBA.Any) return Ref;
        function To_Ref (From : CORBA.Object.Ref'CLASS) return Ref;

    endElement;
    -- …
end ASIS_CORBA;

with CORBA.Object; with ASIS_CORBA.Element;
package ASIS_CORBA.Generics is -- generic portion of Asis.Iterator.Traverse_Control

    type Ref is new CORBA.Object.Ref with null record;   -- abstract
    -- Narrow/Widen functions
    function To_Ref (From : CORBA.Any) return Ref;
    function To_Ref (From : CORBA.Object.Ref'CLASS) return Ref;

    procedure Pre_Operation
        (Self           : Ref;
         This_Element   : in     ASIS_CORBA.Element.Ref;
         What_Next      : in out ASIS_CORBA.Traverse_Control);
```

```
    procedure Post_Operation
         (Self          : Ref;
          This_Element  : in     ASIS_CORBA.Element.Ref;
          What_Next     : in out ASIS_CORBA.Traverse_Control);
          -- no state info needed, children will add state
end ASIS_CORBA.Generics

with CORBA.Object; with ASIS_CORBA.Element; with ASIS_CORBA.Generics;
package ASIS_CORBA.Iterator is

    type Ref is new CORBA.Object.Ref with null record;
    -- Narrow/Widen functions
    function To_Ref (From : CORBA.Any) return Ref;
    function To_Ref (From : CORBA.Object.Ref'CLASS) return Ref;

    procedure Traverse_Element
       (Self          : Ref;
        Starting_From : in     ASIS_CORBA.Element.Ref;
        What_Next     : in out ASIS_CORBA.Traverse_Control;
        Using         : in     ASIS_CORBA.Generics.Ref);
        -- Using contains state and implementations of My_Pre_Operation and
        -- My_Post_Operation.
end ASIS_CORBA.Iterator;
```

Annex D
(informative)
Rationale

D.1 Benefits of code analysis

Why is something like ASIS needed? ASIS provides a basis for implementing portable tools that are aimed at analyzing static properties in Ada source code. Such code analysis capability has in general been under leveraged in software organizations; but for the Ada language in particular, it can greatly enhance the development process. Code analysis automation can harness the excellent software engineering features of Ada to facilitate code comprehension, high reliability, and high quality of the software product. The following text presents some motivational background.

D.1.1 Definition

Code analysis is the inspection of software source code for the purpose of extracting information about the software. Such information can pertain to individual software elements (e.g., standards compliance, test coverage), the element attributes (e.g., quality, correctness, size, metrics), and element relationships (e.g., complexity, dependencies, data usage, call trees); thus, it can support documentation generation, code review, maintainability assessment, reverse engineering, and other software development activities.

Extracted information falls into two major categories: *descriptive* reports which present some view of the software without judgment (e.g., dependency trees, call trees), and *proscriptive* reports which look for particular deficiencies — or their absence (e.g., stack overflow, excessive complexity, unintended recursion).

D.1.2 Applicability

Broadly speaking, the application of automated code analysis in the software development process promises, among other things, to:

- Promote ***discipline and consistency*** during development, increasing productivity and reducing unintended variation.
- Provide empirical evidence and ***metrics*** for process monitoring and improvement.
- Supplement code ***inspection and review***, diversifying beyond the limitations of testing or manual checking.
- Preserve ***architectural integrity*** in the software as compromises are made during development.
- Avoid violations of ***coding standards***, such as the use of inefficient language constructs.
- Increase the ***correctness and quality*** of delivered software, reducing defects via comprehensive assessment.
- Enhance ***safety and security*** by applying formal methods to verify assertions in program code.
- Expedite ***program comprehension*** during maintenance, for engineers new to the code.
- Support ***reengineering and reuse*** of legacy code, reducing costs.
- Result in ***reduced risk*** to budget and schedule.

The software development life-cycle phases where code analysis can be beneficially applied include all those in which source code exists: preliminary design (software architecture and interface definition), through testing and system integration, to maintenance and reengineering. Hence, automated code analysis is a technology that primarily supports the back end of the software life cycle.

D.1.3 Motivation

Over the years, a wide-ranging set of commercial code analysis tools has become increasingly available [11]; examples of such tools include:

- Data flow analysis and usage metrics
- Invocation (call) trees and cross-reference
- Dependency trees and impact analysis
- Timing and sizing estimation
- Test-case generation and coverage analysis
- Usage counts of language constructs
- Quality assessment metrics
- Coding style and standards compliance
- Safety and security verification
- Code browsing and navigation
- Documentation generation
- Reverse engineering and re-engineering
- Language translation and code restructuring

Unfortunately, the current state-of-the-practice in software development either omits code analysis support altogether or only incorporates it as an ancillary, undisciplined, ad hoc resource. For example, it is not uncommon to find within a given project various home-grown tools that support the above goals but which are not recognized as overtly participating in the development process. Such tools can be quite obtuse (very indirect extraction of information) and are typically incomplete (handling only a subset of the development language). Further, they tend to be project-specific (or even person-specific), and cannot be reused in another project: they are later redeveloped from scratch.

These observations corroborate that the need for code analysis is genuine, and that a common set of uniform tools could provide significant benefits to projects. But in the case of Ada software, commercial code analysis tools have historically proven to be barely adequate, manifesting a variety of problems whose nature and origin are described below.

D.2 Technology for code analysis

For Ada, why is ASIS the best approach? Code analysis tools are not new, having been available for decades; but the advent of the Ada language has exposed a variety of analysis limitations and has consequently demanded more comprehensive technology. The following text articulates various Ada-specific issues from a historical perspective: it reviews several technologies that have historically been applied — with varying degrees of success — to code analysis specifically targeted to Ada software. The review is not comprehensive, but it sketches the evolution of issues that have propelled the development of the ASIS concept.

D.2.1 Code parsers

Historically, many commercial code analysis tools have been supplied by compiler vendors in conjunction with their compiler products. But as the community of CASE tool vendors has grown, such tools are often available independent of any compiler. Tool developers have found that conventional parser technology is sufficient for most traditional languages; thus when Ada came along, most vendors expected it would suffice to simply adapt their parsers to handle Ada syntax. But for Ada, the result has held many disappointments:

- Textual code editors are often sensitive to Ada syntax but not to Ada semantics.
- Graphical design editors yield valid graphics, but invalid Ada designs.
- Source-level tools such as debuggers are forced to understand and traverse the internal data structures of program libraries rather than the text of original source files.
- Reverse engineering and test tools manifest difficulties when trying to resolve overloaded subprogram names or renamings.
- Except for compilers, Ada tools do not require Ada Compiler Validation Capability (ACVC) certification; hence, such tools typically fail to handle the complete repertoire of Ada language features.

Consider the case of a toolsmith who wants to develop a call-tree analyzer. For such a tool to accurately process Ada source files, the toolsmith would be forced to build almost the entire front end of an Ada compiler — a decidedly major undertaking that far out-scopes the original tool building effort. But CASE tool vendors are not in the compiler business; most are reluctant to make this major investment, or have tried and failed. Yet tools built on parser technology alone are not able to fully support the semantic richness of Ada.

D.2.2 DIANA

Many Ada compilers store program units into libraries. They typically structure the information according to some proprietary internal form, such as trees of DIANA (Descriptive Intermediate Attributed Notation for Ada — note that the following discussion applies to all internal forms, but that DIANA is singled out due to its public documentation [5, 6]: DIANA had been intended for standardization, but failed due to the unexpectedly wide variation in internal forms). Such trees thus encode both syntactic and semantic information about Ada programs. The root of a DIANA tree corresponds to a compilation unit; the nodes correspond to smaller Ada structures, such as declarations and statements. Node attributes contain descriptive information and references to other nodes.

DIANA trees offer great convenience and power to toolsmiths, and are sufficient to support the implementation of a large variety of tools (including code generators in compilers). For example, with access to DIANA, the toolsmith who wanted to develop the call-tree analyzer would have a fairly straightforward project. Furthermore, the tool would exhibit better performance, bypassing the needless regeneration (and redundant storage) of intermediate compilation results that are already available in the Ada libraries.

The power of DIANA is sufficient to support the implementation of a virtually unlimited variety of tools. In general, any tool that requires the semantic information generated by an Ada compiler can benefit from access to DIANA. But as with any technology, the use of DIANA also has drawbacks:

- A given implementation of DIANA by a vendor is ***subject to change***: upgrades can obsolete tools written against previous versions, hampering maintenance.
- Similarly, DIANA ***implementations*** vary from vendor to vendor: porting a tool across platforms is a risky endeavor.

- DIANA is ***hard to use***: the trees are quite complex, making it difficult to write and debug tools written against a DIANA specification.
- The lack of a simple mapping to Ada makes DIANA ***hard to understand***: as an abstracted representation of an Ada program, it does not map intuitively to concrete Ada structures.
- DIANA is ***not extensible***; but tools may need to add attributes for storing graphical or other tool-specific information.

D.2.3 LRM-interface

Thus a growing need arose to make tool development possible at the Ada level rather than at the internal representation level. It was these issues that drove some Ada compiler vendors to independently develop proprietary higher-level interfaces to encapsulate their Ada program libraries.

In particular, to overcome the drawbacks of DIANA while retaining all of its advantages, Rational Software Corporation developed their LRM-Interface product [8] in the late 1980's. It provided nearly the same power as DIANA, through services that extracted a variety of information from the internal DIANA trees. The LRM-Interface was also considerably easier to understand than DIANA, because it used the already-familiar terminology defined in the Ada LRM (the original Ada Standard, ISO 8652:1987 [12], or its more recent version, the ISO/IEC 8652:1995 [7]). Furthermore, the LRM-Interface was not subject to change (or at least much less so than was the underlying DIANA), so tools written against it were easily migrated to updated implementations.

Regardless of LRM-Interface specifics, this and other similar approaches generally provide great flexibility: for example, an ad hoc tool can be easily and quickly built by in-house engineers, without funding the development or specialization of a commercial tool. But as expected, this approach also has shortcomings:

- While DIANA as a data structure is not extensible, the above interfaces can be extended in the sense of user-supplied secondary functions built on the functions already provided; even so, all the functions are ***read-only*** and cannot modify any state.
- Importing the subject source code into the tool environment can require edits that necessarily result in ***code distortion***, such that original code attributes might not be preserved (e.g., line numbers or the byte sizing of data).
- Tools are ***vendor dependent***, such that a given tool cannot access Ada libraries from multiple vendors, or equivalent tools from multiple vendors cannot access a given Ada library.
- Data interchange is ***not standardized*** among tools, so users can't configure their own integrated toolsets by choosing from competing or complementary vendors.
- Within a software engineering environment (SEE), Ada semantic information remains isolated from, and ***not integrated*** with, other engineering data present in the environment.

D.2.4 ASIS

Historically, only a few Ada vendors provided access to the information contained in their proprietary Ada program libraries, and each such interface was unique. Thus began to emerge the need for an open standard that would allow uniform, vendor-independent access to that information.

Leveraging some informal efforts, the STARS program initiated the development of the Ada Semantic Interface Specification (ASIS) in 1990; but shortly thereafter, the activity became unfunded due to the STARS decision to no longer support standardization efforts. Despite this, several of the involved vendors (primarily TeleSoft) continued the ASIS work on a volunteer

basis. Some time later, Rational also became an active participant and seeded the draft standard by contributing their LRM-Interface specification to ASIS.

In 1992, the Ada Board recognized the potential benefits to the Ada community of an ASIS standard, and recommended that the Ada Joint Program Office (AJPO) director support "by whatever means possible the development of an ASIS standard and its submission to ISO/WG9 for publication." The Association for Computing Machinery (ACM) Special Interest Group on Ada (SIGAda) took on this important work though volunteer effort in the ASIS Working Group (ASISWG) [2]. The ASISWG developed the interface to ISO/IEC 8652:1987. In December 1993, ASIS was viable and the director of the AJPO recommended this interface be used by tools needing information from the Ada program library. The ASISWG then became focused towards developing ASIS for ISO/IEC 8652:1995. As the ASISWG has no standardization authority, an ASIS Rapporteur Group (ASISRG) was established on 28 April 1995 by the ISO/IEC JTC1/SC22 WG9 to standardize ASIS as an International Standard for Ada. ASISWG and ASISRG jointly cooperated to evolve ASIS as an important interface to the Ada compilation environment.

Like its LRM-Interface predecessor, ASIS defines a set of queries and services that provide uniform programmatic access to the syntactic and semantic information contained within Ada library units (i.e., vendor independence). In addition, for each Ada vendor, ASIS clients are shielded from the evolving proprietary details that implement the vendor's library representations and internal forms (i.e., version independence). ASIS is designed for implementation on a variety of machines and operating systems, while also supporting the Ada semantic requirements of a wide range of client tools.

ASIS services are essentially primitive, intended to support higher level, more sophisticated services that address the varied needs of specialized tools. While ASIS currently operates in a read-only mode, it could eventually be extended to support some (probably limited) update capability, enabling client tools to save application-dependent data (e.g., graphical information) within an existing Ada library. Although an ASIS implementor could readily support read-write features, members of the safety-critical community have emphasized the danger of providing a generalized write capability, since this could enable editing of the internal representation to differ from the original source code.

The long-term key is to achieve a critical mass of ASIS implementations. This will promote a new generation of semantically integrated Ada tools, which in turn will increase programmer productivity and decrease Ada development costs. In summary, the availability of ASIS implementations promises to:

- Stimulate ***improved quality*** within existing Ada CASE tools; currently, these tend to be weak in supporting full Ada semantics (e.g., in preserving renamed entities, resolving overloaded subprogram names, etc.).
- Enhance ***safety and security*** by providing for a new class of powerful analysis tools that apply formal methods to verify assertions in program code (e.g., using Pragma Annotate).
- Eliminate the need to import Ada source code into secondary Ada compilation environments, resulting in ***no distortion*** or loss in the subject code (e.g., preserving original line numbers and the byte sizing of data).
- Maximize ***interoperability*** between Ada CASE tools and Ada compilation environments, thus maximizing tool availability.
- Enable the ***data interchange*** of Ada semantic information between complementary or competing Ada CASE tools, thus maximizing user choices for the best capabilities of each.
- Improve the overall ***performance*** of Ada CASE tools, by eliminating the regeneration (and redundant storage) of Ada semantic information that already exists in Ada libraries.
- Facilitate in-house development of informal but powerful ad hoc Ada tools, providing ***flexibility*** as needed without funding Ada CASE vendor specializations.

- Promote standardization in software engineering environments (SEEs), enabling data ***integration*** of Ada semantic information with other engineering data present in the environment.
- Establish ***enabling technology*** for new Ada CASE tools, by eliminating the need for tool vendor investment in proprietary Ada compiler technology; this will have a major impact on stimulating the development of new code analysis capabilities.

D.2.5 Benefits of ASIS standard

As an International Standard, ASIS benefits the Information Technology community by facilitating the development of powerful CASE tools portable amongst the various environments provided by Ada implementors. This portability can only be achieved through the standardization of ASIS at the international level. A standardized ASIS promotes the development of powerful tools for the software engineering environment by providing access to important semantic information otherwise available only through proprietary interfaces. Further, ASIS benefits the Information Technology community as a valuable resource for application development (e.g., decoupling system to system interfaces). The international standardization of ASIS facilitates the use of this important capability in the development of system software applications.

D.3 Design considerations for ASIS

Why has ASIS taken the form that it has? To guide the assessment of design decisions for ASIS, the ASIS Working Group has defined a number of design goals. Most of these embody sound software engineering principles, while others represent a consensus approach regarding controversial issues. The following text discusses some of these.

D.3.1 Design goals

ASIS for ISO 8652:1987, known as ASIS version 1.1.1 [1], was intended to be used as a de facto standard. For ISO/IEC 8652:1995, it was necessary to revise ASIS to take into account the changes in the Ada language definition. When starting the revision of ASIS, the ASIS Working Group agreed upon the following overall design goals: the ASIS design should

- Minimize the cost of converting existing ASIS-based applications to the new Ada standard.
- Be simple and uniform with minimal concepts and minimal operations.
- Allow for standardization at the international level.

The identification of design goals for ASIS was the major accomplishment of the 19-22 April 1994 ASISWG meeting in Boulder, Colorado. These goals were updated at the 27-28 June 1994 ASISWG meeting in Reston, Virginia.

The ASIS design goals have been subdivided into the categories shown below. At a lower level are several design rules identified as important for the interface specification. Also discussed were several potential goals which were felt should not be part of the ASIS interface; these are very useful in providing scope. Several administrative goals were also included.

Design goals (functionality):

- ASIS should provide all semantic information described by the Ada Reference Manual.
- ASIS should not invent new functionality.
- ASIS should provide a read-only interface to the Ada compilation environment.
- ASIS should be able to support Ada semantics for a wide range of tools and application requirements.

- Each ASIS interface should be fully defined and unambiguous.
- ASIS should be easy to learn and easy to use.
- ASIS should use the Ada Standard terminology and style.
- ASIS should have a callable Ada interface.
- ASIS should provide the ability to traverse the full syntactic structure represented in the Ada compilation environment for the original text to the point where the syntax has semantic relevance.
- ASIS should not be misleading, be complete as possible, and provide all information queried (or make obvious to users that they are not getting all the information, and provide an interface to get that additional information).

Design goals (open systems):

- ASIS should be implementor independent.
- ASIS should be non-proprietary.
- ASIS should be designed in such a way that it is implementable by all implementors.
- ASIS should produce semantically consistent results regardless of the implementor.
- ASIS should increase portability of tools across Ada compilers.
- ASIS should make Ada compilation environments Open Systems and Reusable Components for use by other tools.
- ASIS should simplify semantic information retrieval by hiding implementation dependent aspects of Ada environments.

Design goals (migration of ASIS for ISO 8652:1987 to ASIS for ISO/IEC 8652:1995):

- ASIS should ensure that ASIS-based tools are convertible to new ASIS.
- ASIS should minimize the cost of converting ASIS tools.

Design goals (miscellaneous):

- ASIS should be efficient.
- ASIS should be well engineered.
- ASIS documentation should be in ISO format.
- ASIS should allow for standardization at the international level.
- ASIS should be written to not preclude future extensions.
- ASIS should be designed to be simple and uniform with minimal concepts and minimal operations.

Design rules:

- ASIS should be as strongly typed as possible.
- ASIS should declare its own basic types, and rename (or hide) all implementor types and operations, for tool portability.
- ASIS should use exceptions to report failures and errors; ASIS should supplement exceptions with status functions.
- ASIS should allow only ASIS exceptions to escape the interface.
- ASIS should supplement weak compile time type checking of element values by run-time use of ASIS_Inappropriate_Element Exception.
- ASIS should have an existence query for all optional syntax.
- ASIS should be able to report a span for every explicit element.

NOT design goals:

- It is not a goal to support tools having dynamic run-time requirements (e.g., symbolic debugger).
- It is not a goal to define the semantics of the interface in the face of simultaneous library updates.

Administrative goals:

- Resolve issues by consensus rather than simple majority.
- Keep the effort scaled down so it is possible and achievable.

D.3.2 Major changes from ASIS for ISO 8652:1987

The main changes in the ASIS definition are as follows:

- New queries were added for the new language features, such as child units and protected types, and old ones were revised to take into account language changes.
- Some queries of ASIS were aggregated, in order to minimize their total number.
- Whereas ASIS for ISO 8652:1987 was defined as an interface to a program library, ASIS is now an interface to an Ada compilation environment, following the new ideas of Ada about the program structure and the compilation process.
- The set of package specifications defining ASIS was reorganized into a hierarchy of child units.
- Terminology was revised and comments in the ASIS package specifications were reworded to make them consistent with Ada.

D.3.3 Major changes from ASIS for ISO/IEC 15291:1999

The main changes in the ASIS definition are as follows:

- Previous versions of ASIS provided mostly syntactic information regarding an Ada unit or program. An entire new interface and set of packages was added to provide a semantic "view" of units. Routines are also provided to allow for easy transition between these two different views of an Ada program. The semantic view makes it easier to access information that is otherwise implicit or anonymous.
- New queries and types were added to support language features added or revised by Amendment 1 (ISO/IEC 8652:1995/AMD 1:2007) such as new kinds of types, new pragmas, attribute kinds, and use of prefix notation. Existing queries and types were revised to take into account language changes.
- The expected input types and return types were clarified throughout the standard.
- Restricted some past allowed variability between implementations, increasing the portability of source transforming tools.

D.3.4 Essence of Ada and ASIS

There are some fundamental differences between a language definition (i.e., the definition of Ada) and the interface definition of a tool, such as ASIS. Clearly the Ada language definition, as described in the Ada Standard, and ASIS are different in nature and have different aims.

Ada defines the notion of a legal compilation unit, the notion of a legal complete partition and the dynamic semantics of a complete partition. It also defines some aspects of the process of compiling a compilation unit and some properties of the results of this process.

On the other hand, ASIS is more like the requirements definition for a tool which provides information about a set of Ada software components, which may or may not make up a complete partition. Moreover, as a tool, ASIS provides management facilities which go beyond what is

needed in a language definition (e.g., ASIS needs terminology for designating the then-part and the else-part of an if statement).

Despite these differences in nature, the terminology used in Ada and ASIS should be the same whenever possible. Clearly, there should be no contradictions. Closeness in terminology eases understanding of ASIS by those who already know Ada, saves documentation effort, and is justified by improved maintenance, since every correction in Ada can impact ASIS.

Nevertheless, ASIS needs some new terms. Here are some cases:

- To cut down the number of queries, ASIS sometimes needs to aggregate Ada-defined syntactic categories.
- To walk through a heterogeneous data structure, such as a list of compilation units and configuration pragmas, ASIS needs to introduce a unifying concept for item types in the structure.
- Finally, some concepts in Ada are vaguely defined compared to the needs of ASIS. It can then be better to introduce a new name rather than restrict the meaning of the Ada concept. The Ada compilation environment and ASIS context concepts can serve as an example here.

D.4 Major issues regarding ASIS

How has the Ada language revision affected ASIS? The following text presents a discussion on a variety of issues that have arisen in making this transition.

D.4.1 Ada environment and compilation units

One of the major points in the revision of ASIS was the replacement of the idea of an interface to an Ada program library in ASIS 1.1.1 by an interface to an Ada compilation environment. This change has in fact some farther-reaching consequences than it seems at a first look.

The Ada Standard 10.1.4(1) says: "Each compilation unit submitted to the compiler is compiled in the context of an environment declarative_part (or simply, an environment), which is a conceptual declarative_part that forms the outermost declarative region of the context of any compilation. At run time, an environment forms the declarative_part of the body of the environment task of a partition". And the next paragraph says: "The declarative_items of the environment are library_items appearing in an order such that there are no forward semantic dependencies." Ada Standard 10.1.4(5) then says: "When a compilation unit is compiled, all compilation units upon which it depends semantically shall already exist in the environment;..."

At a first look, there seems to be a contradiction between Ada Standard 10.1.4(1,2) which defines an environment as a declarative_part and says that an environment contains library_items, but not compilation_units, and Reference Manual 10.1.4(5) which speaks about the existence of compilation units in the environment.

But this apparent "contradiction" is resolved in Ada Standard 10.1.1(9), which says: "The term compilation unit is used to refer to a compilation_unit. When the meaning is clear from context, the term is also used to refer to the library_item of a compilation_unit or to the proper_body of a subunit (that is, the compilation_unit without the context_clause and without the separate (parent_unit_name))."

This means the Ada Standard defines compilation_unit as a syntactical category, and at the same time it defines "compilation unit" (without underscore and typed in regular font) as another technical term, which can denote either one of the syntactical categories "compilation_unit", "library_item" or "proper_body". ASIS is aware of this subtle difference and consistently use the term "compilation unit". In ASIS "Compilation_Unit" is the name of a type. Objects of this type are always compilation_units (except if they are nonexistent, unknown, or null).

The purpose of the Ada concept of an environment is threefold. It allows Ada to define what is a legal compilation unit, what is a legal partition and what is the behavior of a partition at run-time. Strictly speaking, an Ada environment contains only library_items; indeed, context clauses do not make sense inside of a declarative part, and subunits are enclosed in their parent unit in place of the corresponding stub.

D.4.2 ASIS context and inconsistency

ASIS has to deal with sets of compilation units. To avoid any confusion, it therefore introduces a new term, the concept of a context, which stands for a set of compilation_units maintained by an Ada compilation system. A context has a state, consisting of the compilation units belonging to it, and the configuration pragmas applying to them, but also depending on the history of compiler runs, or other actions taken by the compilation system. Thus, the ASIS concept of a context resembles closely an Ada environment when used as a dynamic concept like in the Ada Standard 10.1.4(3), which says: "The mechanisms for creating an environment and for adding and replacing compilation units within an environment are implementation-defined."

Ada does not deal with arbitrary sets of compilation units as a whole, but ASIS does.

As an example, suppose we first compile a package P, then a procedure A withing P, then we modify P, recompile it, and finally compile the procedure B which also is withing P; we also suppose that all compiler runs are successful (i.e., do not detect compilation errors):

```
compile
   package P is ...
compile
   with P;
   procedure A ...

edit
   package P and change it to P'

compile  -- reference to P'
   package P is ...
compile
   with P;  -- reference to P'
   procedure B ...
```

As far as the legality of compilation units is the concern, the environments can be distinguished as: P alone, P together with A, P' alone, and P' together with B. It could well be that compiling A with P' yields a compilation error. Nevertheless, all these environments contain only consistent unit sets following Ada Standard 10.1.4 (5), which says: "When a compilation unit is compiled, all compilation units upon which it depends semantically shall already exist in the environment; the set of these compilation units shall be consistent in the sense that the new compilation unit shall not semantically depend (directly or indirectly) on two different versions of the same compilation unit, nor on an earlier version of itself."

Only when the results of these compilations are used to build a partition, the inconsistency of depending on two versions for P can be detected by the implementation following Ada Standard 10.2 (27), which says: "The implementation shall ensure that all compilation units included in a partition are consistent with one another, and are legal according to the rules of the language."

As a conclusion, we can say that Ada does not deal with the above sequence of compiler runs as a whole, and especially does not bother about possible "inconsistencies".

On the contrary, it is clearly desirable for ASIS to deal with the previously described situation as a whole, so that tools can be built capable of analyzing dependencies between compilation units and possible inconsistencies in the state of the "context". It is the idea that an ASIS context can refer to such a situation in some implementation-dependent way. As defined now, in no case ASIS will provide access to both versions of P, even indirectly by references from A and B. Depending on the ASIS implementation, the compilation unit A may or may not be part of a context. If it is, then ASIS reports that A is obsolete (e.g., by saying that A is inconsistent with P).

If issued, queries about A can raise exceptions or produce wrong results. This does not preclude some ASIS implementation providing correct answers to structural queries about A (which are about syntax only).

As we have just seen, ASIS introduces the idea of inconsistencies within a context. An inconsistency is always expressed in terms of pairs of compilation units, the second depending on the first: (A depends on P). An inconsistency is reported if the latest compiler runs for A and P used different versions of P, or different versions of any other unit upon which P depends.

This is perhaps the place to speak about the concept of a version of a compilation unit. Even though Ada uses the term in Ada Standard 10.1.4 (5), it does not define it. On the contrary, ASIS does not use it at all. ASIS provides queries for retrieving a library unit having a given name; it is stated that at most one unit can be retrieved, which means that library units, but also compilation units, are uniquely identified by their fully expanded names, and that the concept of a version does not make sense in ASIS.

The Ada environment and ASIS context concepts, though similar, are nevertheless different, and the same holds for the concept of consistency. However, and even if technically perhaps not entirely correct, the idea of ASIS being an interface to an Ada compilation environment reflects intuitively quite well the very nature of ASIS.

D.4.3 Implicit declarations

Sometimes a concept is missing as such in ASIS, but it can be rebuilt indirectly (e.g., implicit declarations of a declarative region can only be retrieved indirectly). Ada distinguishes between explicit and implicit declarations (Ada Standard 3.1 (5)): "A declaration is a language construct that associates a name with (a view of) an entity. A declaration can appear explicitly in the program text (an explicit declaration), or can be supposed to occur at a given place in the text as consequence of the semantics of another construct (an implicit declaration)." In many ways, Ada deals in an uniform way with explicit and implicit declarations. We will see that this is not true for ASIS.

ASIS defines several queries returning lists of explicit declarations from different kinds of declarative regions (i.e., Body_Declarative_Items, Visible_Part_Declarative_Items, Private_Part_Declarative_Items, and Block_Declarative_Items).

ASIS has several approaches to deal with implicit declarations:

a) A first case is **type declarations**. To retrieve the implicit declarations associated with a type, ASIS provides queries to get all the predefined and inherited subprograms associated with it, and also its implicit components, when it is a record.

b) A second case is **statement_identifiers**. Labels, block names and loop names are statement_identifiers. Ada Standard 5.1(12) says that "For each statement_identifier, there is an implicit declaration (with the specified identifier) at the end of the declarative_part of the innermost block_statement or body that encloses the Statement_Identifier." ASIS does not provide a query listing the statement_identifiers implicitly declared at the end of a declarative_part. The only way to find out about statement_identifiers is to access the statements and then to retrieve the identifiers attached to them by means of the queries Label_Names and Statement_Identifier.

The two cases show that it is not easy for an ASIS-based application to retrieve all implicit declarations of a given declarative region.

Nevertheless, the approach taken by ASIS is justifiable because compilers usually do not store the implicit statement_identifier declarations in their intermediate representations, and it would therefore be a major burden for an ASIS implementation to yield them as a result of a query.

D.4.4 Abstract "=" for private types

An Is_Equal function has been defined for the private types Context (Clause 3.3), Element (Clause 3.4), Compilation_Unit (Clause 3.8), Container (Clause 9.1), Line (Clause 20.1), Id (Clause 21.1), Record_Component (Clause 22.1), and Array_Component (Clause 22.3). An abstract function "=" has been defined for each private type because the semantics for the Is_Equal function for these types may differ from the semantics of the default "=" operator. This prevents improper use of the "=" operator.

D.4.5 Usage names and expressions

To minimize the number of queries, ASIS sometimes unifies several concepts. For usage names, Ada makes a difference between names which are and which are not expressions. ASIS considers all usage names as expressions.

For instance, the name of the procedure in a procedure call statement is considered to be an expression. Only when trying to apply to such an "expression" a query specific to an expression, yielding for example the type of the expression, ASIS will say that it is not an expression by returning a Nil_Element.

This approach allows ASIS to have a smaller element kinds classification. ASIS for ISO 8652:1987 already worked this way, and compatibility is another reason for keeping the approach.

D.4.6 Select alternative

Let's start by recalling the Ada syntax of a select_alternative (Ada Standard 9.7.1 (4-7)):

```
select_alternative ::= accept_alternative
                     | delay_alternative
                     | terminate_alternative

accept_alternative ::= accept_statement [sequence_of_statements]

delay_alternative ::= delay_statement [sequence_of_statements]

terminate_alternative ::= terminate;
```

When analyzing a selective_accept, ASIS sees the select_alternative just as a sequence_of_statements, and does not distinguish between its three possible forms. The advantage of this approach shows up when the ASIS-application programmer needs to deal with statements in a uniform way, for instance in a recursive descent. The price to pay is that she has to do her own analysis when the distinction is needed, by examining the first statement in the sequence.

D.4.7 Attribute definition clauses

In Ada Standard 13.3 (2), Ada says that an attribute_definition_clause is defined as:

```
attribute_definition_clause ::=
            for local_name'attribute_designator use expression;
          | for local_name'attribute_designator use name;
```

In Ada Standard 4.1, Ada says that a name may be an attribute_reference, which then is formally defined in Ada Standard 4.1.4 (2) by:

```
attribute_reference ::= prefix'attribute_designator
```

ASIS uses this latter definition to simplify the attribute_definition_clause definition:

```
attribute_definition_clause ::=
            for attribute_reference use expression;
          | for attribute_reference use name;
```

From an attribute_definition_clause, ASIS therefore retrieves an attribute_reference, which can be further decomposed into the prefix and the attribute_designator parts.

D.4.8 Configuration pragmas

According to Ada Standard 10.1.5 (8), the purpose of configuration pragmas is: "They are generally used to select a partition-wide or system-wide option." The same paragraph says about their placement that "[T]hey shall appear before the first compilation_unit of a compilation.", and later on about their "scope" that "The pragma applies to all compilation_units appearing in the compilation, unless there are none, in which case it applies to all future compilation_units compiled into the same environment".

ASIS does not use the notion of a compilation ordering. The first reason for this is to keep ASIS as small as possible. The second justification is that compilers usually do not keep track of compilations, but only of their effects.

However, some ASIS-based applications could need information about configuration pragmas. ASIS therefore introduces a special kind of compilation unit, called A_Configuration_Compilation, containing only configuration pragmas. By retrieving the content of A_Configuration_Compilation, it is possible to know the configuration pragmas which are in effect for the next compiler run within this context.

It is important to notice here that ASIS provides also a query to retrieve all the pragmas which were in effect when a given compilation unit was compiled, including both configuration pragmas and pragmas specific to the unit; clearly, some of the former could differ from those found currently in A_Configuration_Compilation.

D.4.9 Queries with additional context parameter

Several queries, including Corresponding_Declaration and Corresponding_Body, are overloaded with an additional parameter, Context. The additional queries are there for flexibility and power, if an ASIS implementation can take advantage of it.

Some ASIS implementations may be able to support multiple open Contexts at the same time. Tools that make use of more than one Context could be severely limited if they do not have the control provided by the Context parameter. An example of such a tool is one that compared two contexts, identifying units appearing in both and ensuring their semantic dependent units were consistent, perhaps for some configuration management purpose.

The Corresponding_Declaration in a different context should always be the declaration upon which the body depends semantically in that context, not just an unrelated declaration of the same name.

ASIS implementations that only allow one open Context can implement these pairs of queries in tandem, with one simply calling the other and providing the current Context as a parameter.

D.4.10 Ids

Ids provide a mechanism to refer to the same ASIS element from one tool analysis to a separate tool analysis. The type Id is a way to identify a particular element (i.e., a unique reference to an Element) which is efficient and persistent as long as the environment is not recompiled. Ids allow a tool to create a database of visited elements to support a variety of tool requirements. An Id can be passed from one ASIS analysis tool to another, facilitating the sharing of analysis thus eliminating the requirement to recompute the analysis for the same element. It could also be used to generate hyper-linked analysis reports and support the pipelining of analysis amongst integrated tools. Ids can be written to files. Ids can be read from files and converted into an Element value with the use of a suitable open Context.

D.4.11 Data decomposition

The optional Data_Decomposition package was developed to support the special needs of applications in decoupling system software interfaces. Such interfaces could support communications between two platforms or communications between two subsystems within a system. Typically an interface between two software systems is accomplished through a rigorously defined interface. This rigorous interface is beneficial since it defines the expected communications protocol between the two computers. However, this interface is a serious impediment to upgrading a system when only part of the system is to be upgraded. The new system typically requires a new interface to support the new capabilities. Hence software on both sides of the interface needs to be upgraded to enhance the system.

Data_Decomposition allows the upgrade of part of a critical system without changing software in the rest of the system. Hence it provides significant flexibility for the life-cycle of a system. Decoupling the interface between two software systems can be accomplished when each system has access to semantic information in the other system's Ada compilation environment. This semantic information describes objects sent across the interface and could serve as a means to identify its operations. Data_Decomposition was specifically developed to support a one-way interface between a United States Air Force satellite and the ground station's message analysis (delog) processing.

The initial delogger implementation required a rigorous interface protocol and specialized processing for each possible message. This was expensive from a development perspective. The time required to build the "types dictionary" took 704 hours for each build, of which there were many.

Their practical solution was to develop a generalized processor, capable of interpreting any message type sent across the interface. This required knowledge of the semantics of data in the message. This *Universal Delogger* was built using ASIS to analyze messages sent for data analysis [13]. The process was:

a) Implement both the satellite software and the ground software in Ada;

b) Send each message from the satellite to the ground station as a byte-stream of application data with a header: the data based on an Ada record type definition of heterogeneous composite types; and the header an identifier for this Ada record definition.

c) Use the ASIS interface to obtain the Ada record type information associated with the message header.

d) Use the ASIS interface to map (decompose) the byte-stream into the appropriate Ada record type.

e) Decode the message for analysts and analytical processing based on the composite types in the Ada record. The named components with their values are then presented to an operator.

Using this approach, the ground station can perform generalized processing for any message from any satellite providing it includes an ASIS interface to the Ada compilation environment that developed the satellite software. Developmental cost benefit: Many, but most notable: each major build had to rebuild the message dictionary; the time required to build this "types dictionary" was reduced from 704 hours to 2 hours for each rebuild. Also notable is the ground station can process messages from newer satellite systems without any changes in the software on the ground station, even though message content is changed.

D.5 Conclusion

The Ada language definition and ASIS have different aims. When taking this into account, it can be said that ASIS follows, overall, as much as possible the syntax structure of Ada; that it uses Ada-defined terms consistently; and it does not introduce unnecessarily new terms and concepts. Clearly, ASIS has a need for unifying concepts (e.g., for traversing heterogeneous lists), and it introduces them quite carefully and advisedly. Considering that ASIS users and ASIS implementors are both interested in achieving stability of the ASIS definition as soon as possible, and that ASIS has already proved its effectiveness for developing useful tools, the ASIS Working Group and the ASIS Rapporteur Group believe the ASIS definition is now ready for standardization.

D.6 Acronyms

ACM	Association for Computing Machinery
ACVC	Ada Compiler Validation Capability
AJPO	Ada Joint Program Office
ANSI	American National Standards Institute
API	Application Program Interface
APSE	Ada Programming Support Environment
ASCII	American National Standard Code for Information Interchange
ASIS	Ada Semantic Interface Specification
ASISRG	ASIS Rapporteur Group
ASISWG	ASIS Working Group
CASE	Computer-Aided Software Engineering
CORBA	Common Object Request Broker Architecture (OMG)
COTS	Commercial, Off-The-Shelf
CPU	Central Processing Unit
DIANA	Descriptive Intermediate Attributed Notation for Ada
DII	Dynamic Interface Invocation
IDL	Interface Definition Language
IEC	International Electrotechnical Commission
ISO	International Standards Organization
LRM	Language Reference Manual (Ada)
OMG	Object Management Group
ORB	Object Request Broker
SEE	Software Engineering Environment
SIGAda	Special Interest Group on Ada
STARS	Software Technology for Adaptable, Reliable Systems

Annex E
(informative)
Summary of ASIS Entities

This annex lists the entities defined by ASIS.

E.1 ASIS Defined Packages

This clause lists all ASIS defined packages.

Access_Views
  child of Asis.Object_Views 23.10
Ada_Environments
  child of Asis 8
Asis 3
Callable_Holders
  in Asis.Callable_Views 23.9.7
Callable_Views
  child of Asis 23.9
Clauses
  child of Asis 19
Compilation_Units
  child of Asis 10
Composite
  child of Asis.Subtype_Views 23.8
Containers
  child of Asis.Ada_Environments 9
Data_Decomposition
  child of Asis 22
Declarations
  child of Asis 15
Declarative_Regions
  in Asis.Views 23.2.4
Definitions
  child of Asis 16
Elementary
  child of Asis.Subtype_Views 23.7
Elements
  child of Asis 13
Errors
  child of Asis 4
Exception_Holders
  in Asis.Exception_Views 23.13.2
Exception_Views
  child of Asis 23.13
Exceptions
  child of Asis 5
Expressions
  child of Asis 17
Generic_Holders
  in Asis.Generic_Views 23.12.7
Generic_Views
  child of Asis 23.12
Ids
  child of Asis 21
Implementation
  child of Asis 6
Iterator
  child of Asis 14
Object_Holders
  in Asis.Object_Views 23.5.6
Object_Views
  child of Asis 23.5
Package_Holders
  in Asis.Package_Views 23.11.9
Package_Views
  child of Asis 23.11
Permissions
  child of Asis.Implementation 7
Profiles
  child of Asis 23.6
Program_Unit_Vectors
  in Asis.Program_Units 23.3.4
Program_Units
  child of Asis 23.3
Relations
  child of Asis.Compilation_Units 12
Statement_Holders
  in Asis.Statement_Views 23.14.3
Statement_Views
  child of Asis 23.14
Statements
  child of Asis 18
Subtype_Holders
  in Asis.Subtype_Views 23.4.7
Subtype_Vectors
  in Asis.Subtype_Views 23.4.8
Subtype_Views
  child of Asis 23.4
Tagged_Subtype_Vectors
  in Asis.Subtype_Views.Composite 23.8.4
Text
  child of Asis 20
Times
  child of Asis.Compilation_Units 11
View_Declaration_Vectors
  in Asis.Views 23.2.10
View_Holders
  in Asis.Views 23.2.13
View_Vectors
  in Asis.Views 23.2.14
Views
  child of Asis 23.2
  child of Asis.Declarations 23.15
  child of Asis.Definitions 23.16
  child of Asis.Expressions 23.17

E.2 ASIS Defined Types and Subtypes

This clause lists all ASIS defined types and subtypes.

A_Body_Stub *subtype of* Declaration_Kinds
  in Asis 3.7.4
A_Formal_Declaration *subtype of* Declaration_Kinds
  in Asis 3.7.4
A_Full_Type_Declaration *subtype of* Declaration_Kinds
  in Asis 3.7.4
A_Generic_Declaration *subtype of* Declaration_Kinds
  in Asis 3.7.4
A_Generic_Instantiation *subtype of* Declaration_Kinds
  in Asis 3.7.4
A_Generic_Renaming *subtype of* Unit_Kinds
  in Asis 3.10.1
A_Generic_Unit_Declaration *subtype of* Unit_Kinds
  in Asis 3.10.1
A_Generic_Unit_Instance *subtype of* Unit_Kinds
  in Asis 3.10.1
A_Library_Unit_Body *subtype of* Unit_Kinds
  in Asis 3.10.1
A_Number_Declaration *subtype of* Declaration_Kinds
  in Asis 3.7.4
A_Renaming *subtype of* Unit_Kinds
  in Asis 3.10.1
A_Renaming_Declaration *subtype of* Declaration_Kinds
  in Asis 3.7.4
A_Subprogram_Body *subtype of* Unit_Kinds
  in Asis 3.10.1
A_Subprogram_Declaration *subtype of* Unit_Kinds
  in Asis 3.10.1
A_Subprogram_Renaming *subtype of* Unit_Kinds
  in Asis 3.10.1
A_Subunit *subtype of* Unit_Kinds
  in Asis 3.10.1
A_Type_Declaration *subtype of* Declaration_Kinds
  in Asis 3.7.4
Access_Definition_Kinds
  in Asis 3.7.13
Access_Object_View
  in Asis.Object_Views.Access_Views 23.10.1
Access_Subtype
  in Asis.Subtype_Views.Elementary 23.7.4
Access_To_Object_Definition *subtype of*
  in Asis 3.7.12
Access_To_Object_Subtype
  in Asis.Subtype_Views.Elementary 23.7.5
Access_To_Subprogram_Definition *subtype of*
  in Asis 3.7.12
Access_To_Subprogram_Subtype
  in Asis.Subtype_Views.Elementary 23.7.6
Access_Type_Definition *subtype of* Element
  in Asis 3.6
Access_Type_Kinds
  in Asis 3.7.12
An_Anonymous_Access_to_Object_Definition *subtype of*
  in Asis 3.7.13
An_Anonymous_Access_to_Subprogram_Definition
    subtype of
  in Asis 3.7.13
An_Object_Declaration *subtype of* Declaration_Kinds
  in Asis 3.7.4
Array_Component
  in Asis.Data_Decomposition 22.3
Array_Component_Iterator
  in Asis.Data_Decomposition 22.6
Array_Component_List
  in Asis.Data_Decomposition 22.4
Array_Subtype
  in Asis.Subtype_Views.Composite 23.8.2
ASIS_Integer *subtype of*
    Implementation_Defined_Integer_Type
  in Asis 3.1
ASIS_Natural *subtype of* ASIS_Integer
  in Asis 3.1
ASIS_Positive *subtype of* ASIS_Integer
  in Asis 3.1
Aspect_Clause *subtype of* Element
  in Asis 3.6
Aspect_Clause_Kinds
  in Asis 3.7.25
Aspect_Clause_List *subtype of* Element_List
  in Asis 3.6
Association *subtype of* Element
  in Asis 3.6
Association_Kinds
  in Asis 3.7.18
Association_List *subtype of* Element_List
  in Asis 3.6
Attribute_Kinds
  in Asis 3.7.21
Callable_Holder
  in Asis.Callable_Views 23.9.7
Callable_View
  in Asis.Callable_Views 23.9.1
Callable_View_Kinds *subtype of* View_Kinds
  in Asis.Views 23.2.1
Case_Statement_Alternative *subtype of* Element
  in Asis 3.6
Character_Position *subtype of* ASIS_Natural
  in Asis.Text 20.4
Character_Position_Positive *subtype of* Character_Position
  in Asis.Text 20.4
Clause *subtype of* Element
  in Asis 3.6
Clause_Kinds
  in Asis 3.7.24
Compilation_Unit
  in Asis 3.8
Compilation_Unit_List
  in Asis 3.9
Component_Clause *subtype of* Element
  in Asis 3.6
Component_Clause_List *subtype of* Element_List
  in Asis 3.6
Component_Declaration *subtype of* Element
  in Asis 3.6
Component_Definition *subtype of* Element
  in Asis 3.6
Composite_Subtype
  in Asis.Subtype_Views.Composite 23.8.1
Constraint *subtype of* Element
  in Asis 3.6
Constraint_Kinds
  in Asis 3.7.15

E.3 ASIS Defined Subprograms

This clause lists all ASIS defined subprograms.

E.4 ASIS Defined Exceptions

This clause lists all ASIS defined exceptions.

E.5 ASIS Defined Objects

This clause lists all ASIS defined constants, variables, named numbers, and enumeration literals.

Annex F
(normative)
Obsolescent Features

F.1 Annex Contents

This Annex contains descriptions of features of ASIS whose functionality is largely redundant with other features defined by this International Standard. Use of these features is not recommended in newly written programs.

F.2 Obsolescent Features in package Asis

F.2.1 Introduction for Obsolescent Features in package Asis

In addition to the interfaces defined in section 3, the library package Asis also shall provide interfaces equivalent to the obsolescent ones described in the following subclauses.

F.2.2 type Trait_Kinds

This type has been replaced by functions Has_Abstract, Has_Aliased, Has_Limited, Has_Private, Has_Protected, Has_Reverse, Has_Synchronized, Has_Tagged, and Has_Task. Use of the type Trait_Kinds is not recommended in new programs.

Trait_Kinds provide a means of further classifying the syntactic structure or *trait* of certain A_Declaration and A_Definition elements. Trait_Kinds are determined only by the presence (or absence) of certain syntactic constructs. The semantics of an element are not considered.

The syntax of interest here are the reserved words **abstract**, **aliased**, **limited**, **private**, **reverse**, wherever they appear, and the reserved word **access** when it qualifies a definition defining an anonymous type (an access_definition). Trait_Kinds enumerates all combinations useful in this classification.

The subordinate Trait_Kinds allow Declaration_Kinds and Definition_Kinds to enumerate fewer higher level elements, and be less cluttered by all possible permutations of syntactic possibilities. For example, in the case of a record_type_definition, Definition_Kinds can provide just two literals that differentiate between ordinary record types and tagged record types:

```
A_Record_Type_Definition,              -- 3.8(2)   -> Trait_Kinds
A_Tagged_Record_Type_Definition,       -- 3.8(2)   -> Trait_Kinds
```

The remaining classification can be accomplished, if desired, using Trait_Kinds to determine if the definition is abstract, or limited, or both. Without Trait_Kinds, Definition_Kinds needs six literals to identify all the syntactic combinations for a record_type_definition.

Elements expected by the Trait_Kind query are any Declaration_Kinds or Definition_Kinds for which Trait_Kinds is a subordinate kind: the literal definition has "-> Trait_Kinds" following it. For example, the definitions of:

```
A_Discriminant_Specification,          -- 3.7(5)  -> Trait_Kinds
A_Component_Declaration,               -- 3.8(6)
```

indicate A_Discriminant_Specification is an expected kind while A_Component_Declaration is unexpected.

All Declaration_Kinds and Definition_Kinds for which Trait_Kinds is not a subordinate kind, and all other Element_Kinds, are unexpected and are Not_A_Trait.

An_Ordinary_Trait is any expected element whose syntax does not explicitly contain any of the reserved words listed above.

```
type Trait_Kinds is (
   Not_A_Trait,                          -- An unexpected element
   An_Ordinary_Trait,                    -- The declaration or definition does
                                         -- have any of the following traits

   An_Aliased_Trait,                     -- aliased is present
   An_Access_Definition_Trait,           -- The definition defines an anonymous access type
   A_Reverse_Trait,                      -- reverse is present
   A_Private_Trait,                      -- Only private is present
   A_Limited_Trait,                      -- Only limited is present
   A_Limited_Private_Trait,              -- limited and private are present

   An_Abstract_Trait,                    -- Only abstract is present
   An_Abstract_Private_Trait,            -- abstract and private are present
   An_Abstract_Limited_Trait,            -- abstract and limited are present
   An_Abstract_Limited_Private_Trait);   -- abstract, limited, and private are
                                         -- present
```

F.2.3 subtypes Representation_Clause and Representation_Clause_List

Subtypes Representation_Clause and Representation_Clause_List are provided for compatibility with previous editions of this standard. Use of these subtypes is not recommended in new programs.

```
subtype Representation_Clause           is Aspect_Clause;
subtype Representation_Clause_List      is Aspect_Clause_List;
```

F.2.4 function A_Representation_Clause

Function A_Representation_Clause is provided for compatibility with previous editions of this standard. Use of this function is not recommended in new programs.

```
function A_Representation_Clause return Clause_Kinds renames An_Aspect_Clause;
```

F.2.5 type Representation_Clause_Kinds

Subtype Representation_Clause_Kinds and function Not_A_Representation_Clause are provided for compatibility with previous editions of this standard. Use of these entities is not recommended in new programs.

```
   subtype Representation_Clause_Kinds is Aspect_Clause_Kinds;
   function Not_A_Representation_Clause return Aspect_Clause_Kinds renames
Not_An_Aspect_Clause;
```

F.3 Obsolescent Features in package Asis.Implementation.Permissions

F.3.1 Introduction for Obsolescent Features in package Asis.Implementation.Permissions

In addition to the interfaces defined in section 7, the library package Asis.Implementation.Permissions also shall provide interfaces equivalent to the obsolescent ones described in the following subclauses.

F.3.2 function Is_Formal_Parameter_Named_Notation_Supported

This query always returns True. Use of the query Is_Formal_Parameter_Named_Notation_Supported is not recommended in new programs.

```
function Is_Formal_Parameter_Named_Notation_Supported return Boolean;
```

It shall be possible to detect usage of named notation.

F.3.3 function Default_In_Mode_Supported

This query always returns True. Use of the query Default_In_Mode_Supported is not recommended in new programs.

```
function Default_In_Mode_Supported return Boolean;
```

The A_Default_In_Mode kind shall be supported by the implementation.

F.3.4 function Generic_Actual_Part_Normalized

This query always returns False. Use of the query Generic_Actual_Part_Normalized is not recommended in new programs.

```
function Generic_Actual_Part_Normalized return Boolean;
```

The query Generic_Actual_Part shall be able to return both normalized and unnormalized associations.

F.3.5 function Record_Component_Associations_Normalized

This query always returns False. Use of the query Record_Component_Associations_Normalized is not recommended in new programs.

```
function Record_Component_Associations_Normalized return Boolean;
```

The query Record_Component_Associations shall be able to return both normalized and unnormalized associations.

F.3.6 function Is_Prefix_Call_Supported

This query always returns True. Use of the query Is_Prefix_Call_Supported is not recommended in new programs.

```
function Is_Prefix_Call_Supported return Boolean;
```

The ASIS implementation shall have the ability to determine whether calls are in prefix form.

F.3.7 function Function_Call_Parameters_Normalized

This query always returns False. Use of the query Function_Call_Parameters_Normalized is not recommended in new programs.

```
function Function_Call_Parameters_Normalized return Boolean;
```

The query Function_Call_Parameters shall be able to return both normalized and unnormalized associations.

F.3.8 function Call_Statement_Parameters_Normalized

This query always returns False. Use of the query Call_Statement_Parameters_Normalized is not recommended in new programs.

```
function Call_Statement_Parameters_Normalized return Boolean;
```

The query Call_Statement_Parameters shall be able to return both normalized and unnormalized associations.

F.3.9 function Discriminant_Associations_Normalized

This query always returns False. Use of the query Discriminant_Association_Normalized is not recommended in new programs.

```
function Discriminant_Associations_Normalized return Boolean;
```

The query Discriminant_Associations shall be able to return both normalized and unnormalized associations.

F.3.10 function Is_Line_Number_Supported

This query always returns True. Use of the query Is_Line_Number_Supported is not recommended in new programs.

```
function Is_Line_Number_Supported return Boolean;
```

The implementation shall be able to return valid line numbers for Elements.

F.3.11 function Is_Span_Column_Position_Supported

This query always returns True. Use of the query Is_Span_Column_Position_Supported is not recommended in new programs.

```
function Is_Span_Column_Position_Supported return Boolean;
```

The implementation shall be able to return valid character positions for elements.

F.3.12 function Is_Commentary_Supported

This query always returns True. Use of the query Is_Commentary_Supported is not recommended in new programs.

```
function Is_Commentary_Supported return Boolean;
```

The implementation shall be able to return comments.

F.3.13 function Object_Declarations_Normalized

This query always returns False. Use of the query Object_Declarations_Normalized is not recommended in new programs.

```
function Object_Declarations_Normalized return Boolean;
```

The implementation shall not normalize multiple object declarations to an equivalent sequence of single declarations.

F.3.14 function Inherited_Declarations_Supported

This query always returns True. Use of the query Inherited_Declarations_Supported is not recommended in new programs.

```
function Inherited_Declarations_Supported return Boolean;
```

The implementation shall support queries of inherited declarations.

F.3.15 function Inherited_Subprograms_Supported

This query always returns True. Use of the query Inherited_Subprograms_Supported is not recommended in new programs.

```
function Inherited_Subprograms_Supported return Boolean;
```

The implementation shall support queries of inherited subprograms.

F.3.16 function Generic_Macro_Expansion_Supported

This query always returns True. Use of the query Generic_Macro_Expansion_Supported is not recommended in new programs.

```
function Generic_Macro_Expansion_Supported return Boolean;
```

The implementation shall expand generics using macros to support queries.

F.4 Obsolescent Features in package Asis.Elements

F.4.1 Introduction for Obsolescent Features in package Asis.Elements

In addition to the interfaces defined in section 13, the library package Asis.Elements also shall provide interfaces equivalent to the obsolescent ones described in the following subclauses.

F.4.2 function Trait_Kind

This function has been replaced by functions Has_Abstract, Has_Aliased, Has_Limited, Has_Private, Has_Protected, Has_Reverse, Has_Synchronized, Has_Tagged, and Has_Task. Use of the function Trait_Kind is not recommended in new programs.

```
function Trait_Kind (Element : in Asis.Element)
                     return Asis.Trait_Kinds;
```

Element specifies the Element to query.

Returns the Trait_Kinds value of the Element.

Returns Not_A_Trait for any unexpected element such as a Nil_Element, A_Statement, or An_Expression.

Element expects an element that has one of the following Declaration_Kinds:

A_Private_Type_Declaration
A_Private_Extension_Declaration
A_Variable_Declaration
A_Constant_Declaration
A_Deferred_Constant_Declaration
A_Discriminant_Specification
A_Loop_Parameter_Specification
A_Procedure_Declaration
A_Function_Declaration
A_Parameter_Specification

or Element expects an element that has one of the following Definition_Kinds:

A_Component_Definition
A_Private_Type_Definition
A_Tagged_Private_Type_Definition

A_Private_Extension_Definition

or Element expects an element that has one of the following Type_Kinds:

A_Derived_Type_Definition
A_Derived_Record_Extension_Definition
A_Record_Type_Definition
A_Tagged_Record_Type_Definition

or Element expects an element that has one of the following Formal_Type_Kinds:

A_Formal_Private_Type_Definition
A_Formal_Tagged_Private_Type_Definition
A_Formal_Derived_Type_Definition

F.4.3 function Corresponding_Pragmas

This function has been replaced by Corresponding_Aspect_Pragmas. Use of the function Corresponding_Pragmas is not recommended in new programs.

```
function Corresponding_Pragmas (Element : in Asis.Element)
                                return Asis.Pragma_Element_List;
```

Element specifies the element to query.

Returns the list of pragmas semantically associated with the given element, in their order of appearance, or, in any order that does not affect their relative interpretations. These are pragmas that directly affect the given element. For example, a pragma Pack affects the type it names.

Returns a Nil_Element_List if there are no semantically associated pragmas.

If Element is a declaration that includes several defining_names, the result of this query is implementation defined.

Element expects an element that has one of the following Element_Kinds:

A_Declaration
A_Statement

Raises ASIS_Inappropriate_Element with a Status of Value_Error for any element that does not have one of these expected kinds.

Returns an element that has the following Element_Kinds:

A_Pragma

F.4.4 function Representation_Clause_Kind

Function Representation_Clause_Kind is provided for compatibility with previous editions of this standard. Use of this function is not recommended in new programs.

```
function Representation_Clause_Kind
              (Clause : in Asis.Aspect_Clause)
               return Asis.Aspect_Clause_Kinds renames Aspect_Clause_Kind;
```

F.5 Obsolescent Features in package Asis.Declarations

F.5.1 Introduction for Obsolescent Features in package Asis.Declarations

In addition to the interfaces defined in section 15, the library package Asis.Declarations also shall provide interfaces equivalent to the obsolescent ones described in the following subclauses.

F.5.2 function Body_Block_Statement

This function is redundant with the queries Body_Declarative_Items, Body_Statements, and Body_Exception_Handlers. Use of the query Body_Block_Statement is not recommended in new programs.

```
function Body_Block_Statement (Declaration : in Asis.Declaration)
                               return Asis.Statement;
```

Declaration specifies the program unit body to query.

Returns a block statement that is the structural equivalent of the body. The block statement is not Is_Part_Of_Implicit. The block includes the declarative part, the sequence of statements, and any exception handlers.

Declaration expects an element of one of the following Declaration_Kinds:

A_Function_Body_Declaration
A_Procedure_Body_Declaration
A_Package_Body_Declaration
A_Task_Body_Declaration
An_Entry_Body_Declaration

Raises ASIS_Inappropriate_Element with a Status of Value_Error for any element that does not have one of these expected kinds.

Returns an element that has the following Statement_Kinds:

A_Block_Statement

F.5.3 function Object_Declaration_View

This function has been replaced by Object_Declaration_Subtype. Use of the function Object_Declaration_View is not recommended in new programs.

```
function Object_Declaration_View (Declaration : in Asis.Declaration)
              return Asis.Definition;
```

Declaration specifies the declaration element to query.

Returns the definition characteristics that form the view of the object_declaration. The view is the subtype_indication or full type definition of the object_declaration. An initial value, if any, is not part of this view.

For a single_task_declaration or single_protected_declaration, returns the task_definition or protected_definition following the reserved word **is**.

Returns a Nil_Element for a single_task_declaration that has no explicit task_definition.

For a Component_Declaration, returns the Component_Definition following the colon.

For all other object_declaration variables or constants, returns the subtype_indication or array_type_definition following the colon.

Declaration expects an element that has one of the following Declaration_Kinds:

A_Variable_Declaration
A_Constant_Declaration
A_Deferred_Constant_Declaration
A_Single_Protected_Declaration
A_Single_Task_Declaration
A_Component_Declaration
A_Return_Object_Specification

Raises ASIS_Inappropriate_Element with a Status of Value_Error for any element that does not have one of these expected kinds.

Returns an element that has one of the following Definition_Kinds:

Not_A_Definition
A_Subtype_Indication
A_Task_Definition
A_Protected_Definition
A_Component_Definition
A_Type_Definition — the returned element also has the following Type_Kinds:
A_Constrained_Array_Definition

F.5.4 function Declaration_Subtype_Mark

This function has been replaced by Object_Declaration_Subtype. Use of the function Declaration_Subtype_Mark is not recommended in new programs.

```
function Declaration_Subtype_Mark (Declaration : in Asis.Declaration)
                                   return Asis.Name;
```

Declaration specifies the declaration element to query.

Returns the expression element that names the subtype_mark of the declaration.

Declaration expects an element that has one of the following Declaration_Kinds:

A_Discriminant_Specification
A_Parameter_Specification
A_Formal_Object_Declaration
An_Object_Renaming_Declaration

Raises ASIS_Inappropriate_Element with a Status of Value_Error for any element that does not have one of these expected kinds.

Returns an element that has one of the following Expression_Kinds:

An_Identifier
A_Selected_Component
An_Attribute_Reference

F.5.5 function Result_Profile

This function has been replaced by function Result_Subtype. Use of the function Result_Profile is not recommended in new programs.

```
function Result_Profile (Declaration : in Asis.Declaration)
                         return Asis.Name;
```

Declaration specifies the function declaration to query.

Returns the subtype mark expression for the return type for any function declaration.

Declaration expects an element that has one of the following Declaration_Kinds:

A_Function_Declaration
A_Function_Body_Declaration
A_Function_Body_Stub
A_Function_Renaming_Declaration
A_Generic_Function_Declaration
A_Formal_Function_Declaration

Raises ASIS_Inappropriate_Element with a Status of Value_Error for any element that does not have one of these expected kinds.

Returns an element that has one of the following Expression_Kinds:

An_Identifier
A_Selected_Component
An_Attribute_Reference

F.5.6 function Corresponding_Representation_Clauses

This function has been replaced by Corresponding_Aspect_Clauses. Use of the function Corresponding_Representation_Clauses is not recommended in new programs.

```
function Corresponding_Representation_Clauses
            (Declaration : in Asis.Declaration)
            return Asis.Representation_Clause_List;
```

Declaration specifies the declaration to query.

Returns all aspect_clause elements that apply to the declaration, including the aspect clauses for stream-oriented attributes whose prefix is the class-wide type associated with the named entity.

Returns a Nil_Element_List if no clauses apply to the declaration.

If Declaration is a declaration that includes several defining_names, the result of this query is implementation defined.

The clauses returned may be the clauses applying to a parent type if the type is a derived type with no explicit representation. These clauses are not Is_Part_Of_Implicit, they are the aspect_clause elements specified in conjunction with the declaration of the parent type.

Declaration expects an element that has any Declaration_Kinds except Not_A_Declaration.

Raises ASIS_Inappropriate_Element with a Status of Value_Error for any element that does not have one of these expected kinds.

Returns a list of elements that each have the following Clause_Kinds:

A_Representation_Clause

F.6 Obsolescent Features in package Asis.Definitions

F.6.1 Introduction for Obsolescent Features in package Asis.Definitions

In addition to the interfaces defined in section 16, the library package Asis.Definitions also shall provide interfaces equivalent to the obsolescent ones described in the following subclauses.

F.6.2 function Access_To_Function_Result_Profile

This function has been replaced by Access_To_Function_Result_Subtype. Use of the function Access_To_Function_Result_Profile is not recommended in new programs.

```
function Access_To_Function_Result_Profile
               (Type_Definition : in Asis.Type_Definition)
                      return Asis.Name;
```

Type_Definition specifies the Access_Type_Definition to query.

Returns the subtype_mark expression for the return type for the access function.

Type_Definition expects an element of one of the following Type_Kinds:

An_Access_Type_Definition
A_Formal_Access_Type_Definition

that also has one of the following Access_Type_Kinds:

An_Access_To_Function
An_Access_To_Protected_Function

Raises ASIS_Inappropriate_Element with a Status of Value_Error for any element that does not have one of these expected kinds.

Returns an element that has one of the following Expression_Kinds:

An_Identifier
A_Selected_Component

F.6.3 function Corresponding_Parent_Subtype

This function has been replaced by Parent_Subtype_Indication. Use of the function Corresponding_Parent_Subtype is not recommended in new programs.

```
function Corresponding_Parent_Subtype
             (Type_Definition : in Asis.Type_Definition)
                 return Asis.Declaration;
```

Type_Definition specifies the derived_type_definition to query.

Returns the parent subtype declaration of the derived_type_definition. The parent subtype is defined by the parent_subtype_indication. If the parent subtype have no explicit declaration, the value returns is implementation defined.

Type_Definition expects an element that has one of the following Type_Kinds:

A_Derived_Type_Definition
A_Derived_Record_Extension_Definition

Raises ASIS_Inappropriate_Element with a Status of Value_Error for any element that does not have one of these expected kinds.

Returns an element that has one of the following Declaration_Kinds:

An_Ordinary_Type_Declaration
A_Task_Type_Declaration
A_Protected_Type_Declaration
A_Subtype_Declaration
A_Formal_Type_Declaration
An_Incomplete_Type_Declaration
A_Private_Type_Declaration
A_Private_Extension_Declaration

F.7 Obsolescent Features in package Asis.Clauses

F.7.1 Introduction for Obsolescent Features in package Asis.Clauses

In addition to the interfaces defined in section 19, the library package Asis.Clauses also shall provide interfaces equivalent to the obsolescent ones described in the following subclauses.

F.7.2 function Representation_Clause_Name

Function Representation_Clause_Name is provided for compatibility with previous editions of this standard. Use of this function is not recommended in new programs.

```
   function Representation_Clause_Name (Clause : in Asis.Clause)
                                        return Asis.Name renames
Aspect_Clause_Name;
```

F.7.3 function Representation_Clause_Expression

Function Representation_Clause_Expression is provided for compatibility with previous editions of this standard. Use of this function is not recommended in new programs.

```
   function Representation_Clause_Expression (Clause : in Asis.Aspect_Clause)
                                        return Asis.Expression renames
Aspect_Clause_Expression;
```

F.8 Obsolescent Features in package Asis.Data_Decomposition

F.8.1 Introduction for Obsolescent Features in package Asis.Data_Decomposition

In addition to the interfaces defined in 22, the library package Asis.Data_Decomposition (if provided) also shall provide interfaces equivalent to the obsolescent ones described in the following subclauses.

Operations to decompose data values using the ASIS type information and a Portable_Data stream, representing a data value of that type, are provided by this package. These facilities are largely superseded by the Ada streams operations.

An application can write data, using the Asis.Data_Decomposition.Portable_Transfer package to an external medium for later retrieval by another application. The second application reads that data and then uses this package to convert that data into useful information. Simple discrete scalar types can be converted directly into useful information. Composite types, such as records and arrays, shall first be broken into their various discriminants and components.

A data stream representing a record value can be decomposed into a group of discriminant and component data streams by extracting those streams from the record's data stream. This extraction is performed by applying any of the Record_Components which describe the discriminants and components of the record type. Each discriminant and each component of a record type is described by a Record_Component value. Each value encapsulates the information needed, by the implementation, to efficiently extract the associated field from a data stream representing a record value of the correct type.

A data stream representing an array value can be decomposed into a group of component data streams by extracting those streams from the array's data stream. This extraction is performed by applying the single Array_Component which describes the components of the array type. One Array_Component value is used to describe all array components. The value encapsulates the information needed, by the implementation, to efficiently extract any of the array components.

This interface assumes that the data stream is appropriate for the ASIS host machine. For example, the implementation of this interface will not need to worry about byte flipping or reordering of bits caused by movement of data between machine architectures.

All operations in this package will attempt to detect the use of invalid data streams. A data stream is "invalid" if an operation determines that the stream could not possibly represent a value

of the expected variety. Possible errors are: stream is of incorrect length, stream contains bit patterns which are illegal, etc. The exception ASIS_Inappropriate_Element is raised in these cases. The Status value is Data_Error. The Diagnosis string will indicate the kind of error detected.

F.8.2 type Portable_Data

Portable_Data represents an ordered "stream" of data values.

The portable representation for application data is an array of data values. This portable data representation is guaranteed to be valid when written, and later read, on the same machine architecture, using the same implementor's compiler and runtime system. Portability of the data values, across implementations and architectures, is not guaranteed. Some implementors may be able to provide data values which are portable across a larger subset of their supported machine architectures.

Some of the problems encountered when changing architectures are: bit order, byte order, floating point representation, and alignment constraints. Some of the problems encountered when changing runtime systems or implementations are: type representation, optimization, record padding, and other I/O subsystem implementation variations.

The nature of these data values is deliberately unspecified. An implementor will choose a data value type that is suitable for the expected uses of these arrays and data values. Arrays and data values have these uses:

a) Array values are used in conjunction with the Asis.Data_Decomposition interface. The data value type should be readily decomposable, by that package, so that array and record components can be efficiently extracted from a data stream represented by this array type. The efficiency of that interface is a priority.

b) The data value type is read and written by applications. It should have a size that makes efficient I/O possible. Applications can be expected to perform I/O in any or all of these ways:

   1) Ada.Sequential_Io or Ada.Direct_Io could be used to read or write these values.

   2) Individual values may be placed inside other types and those types may be read or written.

   3) The 'Address of a data value, plus the 'Size of the data value type, may be used to perform low level system I/O. Note: This requires the 'Size of the type and the 'Size of a variable of that type to be the same for some implementations.

   4) Individual values may be passed through Unchecked_Conversion in order to obtain a different value type, of the same 'Size, suitable for use with some user I/O facility. This usage is non-portable across implementations.

c) Array values are read and written by applications. The data value type should have a size that makes efficient I/O possible. Applications can be expected to perform I/O in any or all of these ways:

   1) Ada.Sequential_Io or Ada.Direct_Io could be used to read or write a constrained array subtype.

   2) Array values may be placed inside other types and those types may be read and written.

   3) The 'Address of the first array value, plus the 'Length of the array times the 'Size of the values, may be used to perform low level system I/O. Note: This implies that the array type is unpacked, or, that the packed array type has no "padding" (e.g., groups of five 6-bit values packed into 32-bit words with 2 bits of padding every 5 elements).

4) Array values may be passed through Unchecked_Conversion in order to obtain an array value, with a different value type, suitable for use with some user I/O facility. This usage is non-portable across implementations.

The data value type should be chosen so that the 'Address of the first array data value is also the 'Address of the first storage unit containing array data. This is especially necessary for target architectures where the "bit" instructions address bits in the opposite direction as that used by normal machine memory (or array component) indexing. A recommended 'Size is System.Storage_Unit (or a multiple of that size).

Implementations that do not support Unchecked_Conversion of array values, or which do not guarantee that Unchecked_Conversion of array values will always "do the right thing" (convert only the data, and not the dope vector information), should provide warnings in their ASIS documentation that detail possible consequences and work-arounds.

The index range for the Portable_Data type shall be a numeric type whose range is large enough to encompass the Portable_Data representation for all possible runtime data values.

All conversion interfaces always return Portable_Data array values with a 'First of one (1).

The Portable_Value type may be implemented in any way whatsoever. It need not be a numeric type.

```
type Portable_Value is (Implementation_Defined);
subtype Portable_Positive is Asis.ASIS_Positive
   range 1 .. Implementation_Defined_Integer_Constant;
type Portable_Data is array (Portable_Positive range <>) of Portable_Value;
Nil_Portable_Data : Portable_Data (1 .. 0);
```

F.8.3 function Record_Components (stream)

```
function Record_Components
            (Type_Definition : in Asis.Type_Definition;
             Data_Stream     : in Portable_Data)
            return Record_Component_List;
function Record_Components
            (Component   : in Record_Component;
             Data_Stream : in Portable_Data)
            return Record_Component_List;
function Record_Components
            (Component   : in Array_Component;
             Data_Stream : in Portable_Data)
            return Record_Component_List;
```

Type_Definition specifies the record type definition to query. Component specifies a component which has a record subtype, Is_Record(Component) = True. Data_Stream specifies a data stream containing, at least, the complete set of discriminant or index constraints for the type.

These functions are superseded by the Ada Stream operations. The use of any of the functions Record_Components is not recommended in new programs.

Returns a list of the discriminants and components for the indicated record type, using the data stream argument as a guide. The record type shall be either a simple static, or a simple dynamic, record type.

The result describes the locations of the record type's discriminants and all components of the appropriate variant parts. The contents of the list are determined by the discriminant values present in the data stream.

A simple static type will always return the same component list (Is_Equal parts) regardless of the Data_Stream, because the layout of a simple static type does not

change with changes in discriminant values. A simple dynamic type returns different component lists (non-Is_Equal parts) depending on the contents of the Data_Stream, because the contents and layout of a simple dynamic type changes with changes in discriminant values. All return components are intended for use with a data stream representing a value of the indicate record type.

The Data_Stream shall represent a fully discriminated value of the indicated record type. The stream may have been read from a file, it may have been extracted from some enclosing data stream, or it may be an artificial value created by the Construct_Artificial_Data_Stream operation. Only the discriminant portion of the Data_Stream is checked for validity, and, only some discriminant fields may need to be checked, depending on the complexity of the record type. The best approach, for any application that is constructing artificial data streams, is to always provide appropriate values for all discriminant fields. It is not necessary to provide values for non-discriminant fields.

All Is_Record(Component) = True values are appropriate. All return values are valid parameters for all query operations.

Type_Definition expects an element that has the following Element_Kinds:

A_Type_Definition that has one of the following Type_Kinds:
A_Derived_Type_Definition (derived from a record type)
A_Record_Type_Definition

Component expects a component that has one of the following Asis.Data_Decomposition.Type_Model_Kinds:

A_Simple_Static_Model
A_Simple_Dynamic_Model

Raises ASIS_Inappropriate_Element with a Status of Value_Error for any element or component that does not have one of these expected kinds.

NOTE If an Ada implementation uses implementation-dependent record components (Ada Standard 13.5.1 (15)), then each such component of the record type is included in the result.

F.8.4 function Component_Data_Stream

```
function Component_Data_Stream
            (Component   : in Record_Component;
             Data_Stream : in Portable_Data)
            return Portable_Data;

function Component_Data_Stream
            (Component   : in Array_Component;
             Index       : in Asis.ASIS_Positive;
             Data_Stream : in Portable_Data)
            return Portable_Data;

function Component_Data_Stream
            (Component   : in Array_Component;
             Indexes     : in Dimension_Indexes;
             Data_Stream : in Portable_Data)
            return Portable_Data;

function Component_Data_Stream
            (Iterator    : in Array_Component_Iterator;
             Data_Stream : in Portable_Data)
            return Portable_Data;
```

These functions are superseded by the Ada Stream operations. The use of any of the functions Component_Data_Stream is not recommended in new programs.

Component specifies the component or discriminant to be extracted. Index specifies an index, 1..Array_Length, within an array. Indexes specifies a list of index values, there is

one value for each dimension of the array type, each index N is in the range 1..Array_Length (Component, N);. Iterator specifies the array component to extract. Data_Stream specifies the data stream from which to extract the result.

Returns a data stream representing just the value of the chosen Component. The return value is sliced from the data stream. The Data_Stream shall represent a value of the appropriate type. It may have been obtained from a file, extracted from another data stream, or artificially constructed using the Construct_Artificial_Data_Stream operation.

An artificial data stream may raise ASIS_Inappropriate_Element (the Status is Value_Error). Only the constraint values are valid, once they have been properly initialized, and can be safely extracted from an artificial data stream.

Raises ASIS_Inappropriate_Element if given a Nil_Array_Component_Iterator or one where Done(Iterator) = True. The Status value is Data_Error. The Diagnosis string will indicate the kind of error detected.

Component expects any kind of non-Nil component. Raises ASIS_Inappropriate_Element with a Status of Value_Error for any Nil component.

An example:

```
declare
   Component        : Array_Component := ...;
   Iter             : Array_Component_Iterator;
   Array_Stream     : Portable_Data (...) := ...;
   Component_Stream : Portable_Data (...);
begin
   Iter := Array_Iterator (Component);
   while not Done (Iter) loop
      Component_Stream := Component_Data_Stream (Iter, Array_Stream);
      Next (Iter);
   end loop;
end;
```

F.8.5 function Size (stream)

```
function Size (Type_Definition : in Asis.Type_Definition;
               Data_Stream     : in Portable_Data)
              return Asis.ASIS_Natural;
```

This function is superseded by the Ada Stream operations. The use of this function Size is not recommended in new programs.

Type_Definition specifies the type definition to query. Data_Stream specifies a data stream containing, at least, the complete set of discriminant or index constraints for the type.

Returns the 'Size of a value of this type, with these constraints. This is the minimum number of bits that is needed to hold any possible value of the given fully constrained subtype. Only the constraint portion of the Data_Stream is checked.

The Data_Stream may be a data stream or it may be an artificial data stream created by the Construct_Artificial_Data_Stream operation.

Type_Definition expects an element that has the following Element_Kinds:

A_Type_Definition

and Type_Definition expects an element that has one of the following Asis.Data_Decomposition.Type_Model_Kinds:

A_Simple_Static_Model
A_Simple_Dynamic_Model

Raises ASIS_Inappropriate_Element with a Status of Value_Error for any element that does not have one of these expected kinds.

F.8.6 function Portable_Constrained_Subtype

```
generic
    type Constrained_Subtype is private;
function Portable_Constrained_Subtype
            (Data_Stream : in Portable_Data)
            return Constrained_Subtype;
```

This generic function is superseded by the Ada Stream operations. The use of the generic function Portable_Constrained_Subtype is not recommended in new programs.

Data_Stream specifies an extracted component of a record.

Instantiated with an appropriate scalar type, (e.g., Standard.Integer, can be used to convert a data stream to a value that can be directly examined).

Instantiated with a record type, can be used to convert a data stream to a value that can be directly examined.

Instantiations with constrained array subtypes may not convert array values if they were created using the Portable_Array_Type_1, Portable_Array_Type_2, or Portable_Array_Type_3 interfaces.

May raise Constraint_Error if the subtype is a scalar and the converted value is not in the subtype's range.

F.8.7 function Construct_Artifical_Data_Stream

```
function Construct_Artificial_Data_Stream
            (Type_Definition : in Asis.Type_Definition;
             Data_Stream     : in Portable_Data;
             Discriminant    : in Record_Component;
             Value           : in Portable_Data)
            return Portable_Data;
```

This function is superseded by the Ada Stream operations. The use of the function Construct_Artificial_Data_Stream is not recommended in new programs.

Type_Definition specifies the record type definition for the record valued data stream being constructed. Data_Stream specifies the data stream constructed so far; initially specified as the Nil_Portable_Data value. Discriminant specifies the discriminant of the record type that is being set or changed. Value specifies a data stream representing a single discriminant value of the appropriate type.

Used to artificially construct a data stream which represents the discriminant portion of a fully constrained value of the indicated record type. This operation is called once with a value for each discriminant of the record type (the order in which the discriminants are specified is not important). The return value of each call is used as the input Data_Stream for the next.

The resulting artificial data stream may be used solely for the purpose of creating Record_Component values. The values of any non-discriminant fields are arbitrary and quite possibly invalid. The resulting component values may then be used for any purpose. In particular, they may be used to determine First_Bit, Last_Bit, and Size values for all record discriminants and components.

Type_Definition expects an element that has the following Element_Kinds:

A_Type_Definition that has one of the following Type_Kinds:
A_Derived_Type_Definition (derived from a record type)

A_Record_Type_Definition

Raises ASIS_Inappropriate_Element with a Status of Value_Error for any element that does not have one of these expected kinds.

Raises ASIS_Inappropriate_Element, with a Status of Data_Error, if the discriminant Value is inappropriate for the specified Discriminant.

F.9 Obsolescent package Asis.Data_Decomposition.Portable_Transfer

F.9.1 Introduction for Obsolescent package Asis.Data_Decomposition.Portable_Transfer

The library package Asis.Data_Decomposition.Portable_Transfer may exist. If it exists, the package shall provide interfaces equivalent to the obsolescent ones described in the following subclauses.

This package is part of the optional Asis.Data_Decomposition package. It may or may not be present in all ASIS implementations.

It provides support for marshalling and unmarshalling of application data using ASIS.

These interfaces do not make use of the normal ASIS implementation. This is a standalone package. It is not necessary to initialize ASIS before using this interface.

This interface is intended to be separate from the normal ASIS implementation. That is, linking with this interface does not cause the full ASIS implementation to be linked with an application. This keeps the memory cost of this facility to a minimum. However, it also means that the normal ASIS exceptions, Status, and Diagnosis facilities are not available. This interface may propagate exceptions other than those defined in Asis.Errors. In particular, the I/O facilities may raise any of the normal Ada I/O exceptions from package Ada.Io_Exceptions.

F.9.2 generic package Portable_Constrained_Subtype

```
generic
   type Constrained_Subtype is private;
package Portable_Constrained_Subtype is
   function Convert (Value : in Constrained_Subtype)
     return Asis.Data_Decomposition.Portable_Data;
end Portable_Constrained_Subtype;
```

This generic package is superseded by the Ada Stream operations. The use of generic package Portable_Constrained_Subtype is not recommended in new programs.

Value specifies a data value which is to be converted to a portable data stream.

Instantiated with a scalar subtype, or a fully constrained composite subtype, this function will correctly convert values of that subtype to a portable data value. "Correctly" means that no information is lost. No guarantee is made that the result will not be larger (larger 'Size) than the argument.

Instantiation of this generic, with arguments other than scalar types or fully constrained subtypes, will fail to compile or will fail to execute and raise Constraint_Error.

The marshalling and unmarshalling of a non-null access value (either as a top-level object or as a component of an enclosing structure) by means of the Portable_Constrained_Subtype generics is subject to the same restrictions as the predefined Streaming-Oriented attributes for an access type. For most implementations, the bits representing the access value are simply copied and it is the user's responsibility

to ensure the validity (or cope with the non-validity) of those bits when the access value is reconstituted.

F.9.3 generic package Portable_Unconstrained_Record_Type

```
generic
   type Unconstrained_Record_Type (<>) is private;
package Portable_Unconstrained_Record_Type is
   function Convert (Value : in Unconstrained_Record_Type)
                    return Asis.Data_Decomposition.Portable_Data;
end Portable_Unconstrained_Record_Type;
```

This generic package is superseded by the Ada Stream operations. The use of generic package Portable_Unconstrained_Subtype is not recommended in new programs.

Value specifies a record value which is to be converted to a portable data stream.

Instantiated with a record type, this function will correctly convert values of that type to a portable data value. "Correctly" means that no information is lost. No guarantee is made that the result will not be larger (larger 'Size) than the argument.

This generic may also be instantiated with fully constrained subtypes. However, the Portable_Constrained_Subtype generic is likely to be a more efficient mechanism for these subtypes.

Instantiation of this generic, with unconstrained array types, will not fail at compile time or at runtime. However, no guarantee is made that such instantiations will operate correctly; data could be lost.

F.9.4 generic packages Portable_Array_Type_n

```
generic
   type Element_Type is private;
   type Index_Type is (<>);
   type Array_Type is array (Index_Type range <>) of Element_Type;
package Portable_Array_Type_1 is
   function Convert (Value : in Array_Type)
       return Asis.Data_Decomposition.Portable_Data;
end Portable_Array_Type_1;
```

```
generic
   type Element_Type is private;
   type Index_Type_1 is (<>);
   type Index_Type_2 is (<>);
   type Array_Type is array (Index_Type_1 range <>,
                             Index_Type_2 range <>) of Element_Type;
package Portable_Array_Type_2 is
   function Convert (Value : in Array_Type)
      return Asis.Data_Decomposition.Portable_Data;
end Portable_Array_Type_2;
```

```
generic
   type Element_Type is private;
   type Index_Type_1 is (<>);
   type Index_Type_2 is (<>);
   type Index_Type_3 is (<>);
   type Array_Type is array (Index_Type_1 range <>,
                             Index_Type_2 range <>,
                             Index_Type_3 range <>) of Element_Type;
package Portable_Array_Type_3 is
   function Convert (Value : in Array_Type)
       return Asis.Data_Decomposition.Portable_Data;
end Portable_Array_Type_3;
```

These generic packages are superseded by the Ada Stream operations. The use of generic packages Portable_Array_Type_1 Portable_Array_Type_2, and Portable_Array_Type_3 is not recommended in new programs.

Value specifies an array value which is to be converted to a portable data stream.

Instantiated with an unconstrained array subtype, this function will correctly convert values of that subtype to a portable data value. "Correctly" means that no information is lost. No guarantee is made that the result will not be larger (larger 'Size) than the argument. This interface will correctly capture the array index (dope vector) information associated with the array value.

Use the Portable_Constrained_Subtype generic for statically constrained array subtypes.

Array types with more than 3 dimensions can be converted to portable data streams by placing them inside a discriminated record type (where the discriminants are the dynamic array index constraints) and then converting the record value. The record will then contain the necessary array index (dope vector) information.

Bibliography

1. ASISWG/SIGAda; May 1994: Ada Semantic Interface Specification, version 1.1.1, ASIS for ISO 8652:1987.
2. ASISWG/ASISRG World Wide Web site at => http://www.acm.org/sigada/WG/asiswg
3. Colket, C. Barnes, G. Blake, S. Cooper, D. Jorgensen, J. Roby, C. Rittersdorf, D. Rybin, S. Strohmeier, A. Thomas, B.; January/February 1997: "Architecture of ASIS, A Tool to Support Code Analysis of Complex Systems," SIGAda ACM Ada Letters, volume XVII #1, pages 35-40.
4. Ehrenfried, D.; July/August 1994: "Static Analysis of Ada Programs," SIGAda ACM Ada Letters, volume XIV #4, pages 28-35.
5. Goos, W. Wulf, A. Evans, K. Butler; 1983: DIANA: An Intermediate Language for Ada, Springer-Verlag LNCS #161.
6. Intermetrics; 1986: Reference Manual for the Descriptive Intermediate Annotated Notation for Ada (DIANA).
7. ISO/IEC 8652:1995(E), Information technology - Programming languages - Ada.
8. Rational; May 1989: Rational Design and Document Support Tools Guide, document 8043A, version 1.0.
9. Rybin, S. Strohmeier, A. Kuchumov, V. Fofanov, V.; 1996: "ASIS for GNAT: From the Prototype to the Full Implementation," Proceedings of Ada-Europe Conference on Reliable Software Technologies, Springer-Verlag LNCS #1088, pages 298-311.
10. Software Productivity Consortium; 1989: Ada Quality and Style: Guidelines for Professional Programmers, Van Nostrand Reinhold.
11. Software Productivity Consortium; June 1990: "Tools for Static Analysis of Ada Source Code," technical report 90015-N.
12. United States Department of Defense, Ada Joint Program Office; February 83: Reference Manual for the Ada Programming Language, ANSI/MIL-STD-1815A, ISO-8652:1987.
13. White, M.; April 1993: "ASIS for Analysis of Recorded Data," Software Technology Conference (STC'93).

Index

Both ASIS interfaces and Ada syntactic categories are included in this index. In general, ASIS interfaces are denoted with the first letter of each name in uppercase and Ada syntactic categories are denoted in lower case.

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W